CENTRO TECNOLÓGICO
CURSO DE SISTEMAS DA INFORMAÇÃO

Filipe Linemburger
Thiago Mohr da Silveira

Desenvolvimento de ferramenta para o teste de requisições REST

Florianópolis
2018

Filipe Linemburger
Thiago Mohr da Silveira

Desenvolvimento de ferramenta para o teste de requisições REST

Trabalho de conclusão de curso apresentado à Universidade Federal de Santa Catarina para obtenção do grau em Bacharel em Sistemas de Informação.

Orientador: Prof. Leandro José Komosinski

Florianópolis
2018

Filipe Linemburger
Thiago Mohr da Silveira

Desenvolvimento de ferramenta para o teste de requisições REST

Trabalho de conclusão de curso submetido ao Departamento de Informática e Estatística da Universidade Federal de Santa Catarina para a obtenção do Grau de Bacharelado em Sistemas de Informação.

Florianópolis, 01 de dezembro de 2018.

Orientador:

_____

Prof. Leandro José Komosinski
Orientador
Universidade Federal de Santa Catarina

Banca Examinadora:

_____

Leandro da Silva Marques

_____

Prof. Patrícia Vilain

AGRADECIMENTOS

Agradecemos a esta universidade, seu corpo docente, servidores e administração, que viabilizaram nosso crescimento pessoal e profissional. Também ao nosso orientador prof. Dr. Leandro José Komosinski, pelos incentivos e correções necessárias.

Nossas considerações também à banca que foi responsável pela avaliação de nosso TCC, Prof. Patrícia Vilain e Leandro da Silva Marques. Obrigado pelos apontamentos.

Agradecemos também a nossos famíliares e amigos que nos ajudaram e apoiaram nos momentos mais difíceis durante a construção desse projeto.

Por fim, a todos os nossos colegas de graduação que de uma forma ou de outra contribuíram muito para a nossa formação e serão lembrados eternamente.

Muito obrigado a todos.

RESUMO

A qualidade no desenvolvimento dos sistemas é um enorme desafio mediante a alta complexidade dos atuais sistemas desenvolvidos, envolvendo questões humanas, técnicas, de negócio e políticas. No cenário comum de desenvolvimento muitas vezes são feitos testes manuais para verificar se tudo está funcionando conforme a especificação, sendo normal o encontro de defeitos. Esses testes manuais são rápidos e essenciais, mas a execução e repetição de um extenso conjunto de testes manuais é uma tarefa muito onerosa e cansativa. É compreensivo que sejam utilizadas formas para a automatização destes testes.

Este trabalho de conclusão de curso teve como objetivo resolver o problema da repetitividade na grande quantidade de casos de testes que necessitam serem executados em aplicações que utilizam o estilo arquitetural REST (Representational State Transfer - Transferência de Estado Representacional).

A partir deste estudo foi desenvolvido uma ferramenta que possa funcionar com o objetivo de proceder com testes para aplicações REST, que utilizam os métodos GET, POST, PUT, DELETE disponíveis no HTTP para responder às requisições feitas a um URI que deve extrair uma resposta em formato JSON. Esta aplicação é implementada em um serviço WEB na linguagem C Sharp (C#), para a realização de testes de requisições do tipo REST, permitindo assim que qualquer pessoa se cadastre e faça utilização da ferramenta, validando os atributos e a estrutura presentes na resposta das requisições.

A partir da ferramenta desenvolvida foi alcançado como resultado uma série de melhorias no tempo empregado para a realização de testes em uma aplicação REST.

Palavras-chaves: REST, API, serviço WEB, Testes Automatizados, Casos de teste.

ABSTRACT

The Quality in systems development is a huge challenge given the high complexity of current systems, involving human, technical, business and political issues. In the common development scenario manual tests are performed to verify that everything is working according to the specification, being normal the encounter of defects. These manual tests are fast and essential, but running and repeating an extensive set of manual tests is a very burdensome and tiresome task. It is understandable that forms are used for the automation of these tests.

This work aims to solve the problem of repetitiveness in the large number of test cases that need to be executed in applications that use the REST (Representational State Transfer) architectural style.

From this study an application was developed that can work with the objective of proceeding with tests for REST applications that use the GET, POST, PUT, DELETE methods available in HTTP to respond to the requests made to a URI that must extract a response in JSON format. This application is implemented in a WEB service in C Sharp (C #) language, to perform tests of REST type requests, thus allowing anyone to register and make use of the tool, validating the attributes and the structure present in the response of the requests.

From the developed application was achieved as a result a series of improvements in the time spent to perform tests in a REST application.

Key-words REST, API, WEB service, Automated Testing, Test cases.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AJAX - Asynchronous JavaScript e XML

API - Application Programming Interface

C# - C Sharp

DOM - Document Object Model

ECMA - European Computer Manufacturers Association

HTML - Hypertext Markup Language

HTTP - Hypertext Transfer Protocol

ISO - International Organization for Standardization

JSON - JavaScript Object Notation

LINQ - Language Integrated Query

MVC - Modelo, Visualização e Controle

REST - Representational State Transfer

SGBD - Sistema de Gerenciamento de Banco de Dados

SOAP - Simple Object Access Protocol

SQL - Structured Query Language

SWSO - Semantic Web Services Ontology

URL - Uniform Resource Locator

W3C - World Wide Web Consortium

WEB - World Wide Web

WSD - Web Services on Devices

WSDL - Web Services Description Language

XML - Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Com a Revolução Digital, que foi concebida pela disseminação de máquinas (computadores) digitais e do armazenamento de informações de modo digital, deu-se início o período histórico conhecido como Era da Informação. Um dos marcos deste período foi o surgimento e, posteriormente, a ampla adoção da Internet e da World Wide Web - WWW (daqui em diante denominada apenas Web). No início da Web, o conteúdo fornecido era quase que em sua totalidade estático, como um conjunto de documentos a serem acessados em uma determinada URL.

O aparecimento de novas tecnologias e especificações permitiu que os servidores Web fornecessem não apenas conteúdos estáticos, mas também conteúdos dinâmicos, gerados a partir de requisições recebidas via HTTP, dando origem a termos controversos, como: Web 1.0 referente ao período de conteúdos estáticos; Web 2.0, ou colaborativa, dos conteúdos gerados conforme interação com os usuários; e, posteriormente, Web 3.0, ou semântica, "a internet que sabe o que o usuário quer" (MORROW, 2014).

Atualmente vemos a crescente utilização da Internet das Coisas - IoT (sigla em inglês), chamada por alguns autores de Web 4.0, que envolve a comunicação de dispositivos diversos em constante comunicação com a rede, consumindo e originando dados. A Web 5.0 já está em desenvolvimento e envolverá uma maior conectividade, inteligência e utilização dos dados. Os termos são controversos, pois a web já fora concebida com o intuito de ser colaborativa e dinâmica. Então, classificá-la com outros nomes é visto por muitos como sinônimos, ou apelidos, apenas para facilitar a compreensão de certas características da Web (Benito-Osorio, Peris-Ortiz, Armengot, & Colino, 2018).

Nos últimos anos com a crescente utilização de serviços web, foram criados dois tipos de serviços web, que podem ser baseados na arquitetura SOAP (Simple Object Access Protocol) ou em arquitetura REST (Representational State Transfer). O SOAP é um protocolo baseado em XML que permite a troca de mensagens estruturadas com outras aplicações, em geral utiliza meios genéricos para a realização dessa troca, já o REST é uma arquitetura mais flexível que pode ser incorporada a projetos de aplicações distribuídas, quando utilizado para a implementação de serviços web recebe o nome de RESTFUL e utiliza de diversas linguagens de padronização e encapsulamento de dados, cujo transporte é feito por intermédio do protocolo HTTP, tais como JSON, XML e TEXT (Silva, Rocha, & Gonçalves, 2013), assim uma aplicação pode invocar outra para efetuar

tarefas simples ou complexas mesmo que as duas aplicações estejam em diferentes sistemas e escritas em linguagens diferentes. Com o surgimento dessa nova arquitetura de micro serviços, sente-se necessário realizar testes de requisições em serviços REST (Sampaio & Silva, 2014).

Um teste de uma requisição consiste em verificar se a estrutura de dados enviada está de acordo com o definido, erros relacionados à segurança e se o retorno foi ou não implementado para a requisição. No mercado já existem diversas ferramentas para o teste de requisições web que possuem escopo limitado, em grande parte são pagas e não validam campos, como as seguintes:

- Postman
- Newnam
- Apache Jmeter
- SoapUI
- Rest Assured
- Restlet
- RoboHydra
- Gatling

Com o intuito de oferecer uma opção de teste de requisições web de funcione de forma mais simples, ao longo deste trabalho apresentaremos uma ferramenta, suas funcionalidades e seus diferenciais das ferramentas citados.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Este trabalho tem como objetivo projetar um software (projeto de banco de dados, projeto de casos de uso, projeção total da arquitetura de software, desenvolvimento e execução da aplicação WEB), desenvolvido em linguagem de programação C-Sharp (C#) que testa a qualidade dos dados obtidos a partir de disparos de requisições REST. A ideia de construir tal projeto é motivada pela necessidade de haver um software com funções objetivas e práticas para realização de testes em Interface de Programação de Aplicativos (API). De maneira geral, também objetiva-se aprofundar conhecimentos sobre arquiteturas REST e propor uma mais simplificada, objetiva e prática para realização de testes em APIs.

1.1.2 Objetivos Específicos

Para alcançar o objetivo geral, os objetivos específicos deste trabalho são:

1. Facilitar a realização de testes em requisições REST com validação de valor em determinado campo informado pelo usuário.
2. Desenvolver uma aplicação WEB que permita a realização de testes em APIs com requisições REST

1.2 MÉTODO DE PESQUISA

Para que os objetivos deste projeto fossem alcançados, foi adotada uma metodologia de trabalho que consiste nas seguintes etapas:

- Realização de estudo indicativo às tecnologias existentes no que diz respeito ao teste de requisições REST;
- Realizar um estudo referente a linguagens e bibliotecas (APIs) existentes, que podem colaborar para o desenvolvimento do projeto;
- Análise, modelagem e projeção necessária para o desenvolvimento da aplicação.
- Desenvolvimento prático do sistema;
- Publicação da aplicação WEB;

1.2.1 Limites de Pesquisa

Dentro do escopo do trabalho estão delimitados os itens:

- Ênfase na comunicação entre clientes e APIs REST.
- Ênfase no banco de dados SQL Server para linguagem de consulta e inserção dos dados em banco de dados.
- Ênfase na tecnologia JSON para descrição de APIs.
- Ênfase na linguagem de programação C# para desenvolvimento do software e toda a regra de negócio.
- Testes de volume não aplicáveis.
- Não estão incluídos disparos de requisições que necessitem de qualquer tipo de autenticação.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Hypertext Transfer Protocol - HTTP

O Hypertext Transfer Protocol (HTTP), em português Protocolo de Transferência de Hipertexto, é um protocolo para informações distribuídas e hipermídia, utilizado pela Web, baseada em requisições (request) e respostas (responses). O HTTP tem sido usado pela World Wide Web desde 1990. A primeira versão do HTTP, referido como HTTP / 0.9, era um protocolo simples para transferência de dados brutos através da Internet, conforme definido pela RFC por Berners-Lee em 1945 (Fielding R. , 1999).

Na versão HTTP 1.1 foram adicionados os elementos de Status-Code (Código de Status) e Reason Phrase (frase razão). O elemento Código de Status é um código de resultado inteiro de 3 dígitos da tentativa de entender e satisfazer a solicitação. Estes códigos estão totalmente definidos na tabela abaixo (Fielding R. , 1999). Quando o navegador faz uma solicitação de documento a um endereço de um servidor Web, é enviada uma requisição via HTTP e o servidor responde com o mesmo protocolo. Uma requisição pode ter como conteúdo: o método HTTP, a página que será acessada e os parâmetros do formulário, e sua resposta pode ter como conteúdo: o código de status (informa se a solicitação foi realizada com sucesso ou não), o tipo de Conteúdo (HTML, figuras, textos, etc.) e o conteúdo (HTML real, imagem, etc.).
Os códigos de status podem ser as seguintes:

Quadro 1 - Tabela de códigos de status HTTP

Código de status	Frase razão
Informativa	
100	Continuar
101	Mudando Protocolo
102	Processamento (WebDAV)
122	Pedido-URI muito longo

Sucesso	
200	OK
201	Criado
202	Aceito
203	Não-autorizado (desde HTTP/1.1)
204	Nenhum conteúdo
205	Reset
206	Conteúdo parcial
207	Status Multi (WebDAV)
Redirecionamento	
300	Múltipla escolha
301	Movido
302	Encontrado
304	Não modificado
305	Use Proxy (desde HTTP/1.1)
306	Proxy Switch
307	Redirecionamento temporário (desde HTTP/1.1)
Erro do Cliente	
400	Requisição inválida
401	Não autorizado
402	Pagamento necessário

403	Proibido
404	Não encontrado
405	Método não permitido
406	Não aceitável
407	Autenticação de proxy necessária
408	Tempo de requisição esgotou (Timeout)
409	Conflito
410	Gone
411	Comprimento necessário
412	Pré-condição falhou
413	Entidade de solicitação muito grande
414	Pedido-URI Too Long
415	Tipo de mídia não suportado
416	Solicitada de Faixa Não Satisfatória
417	Falha na expectativa
418	Eu sou um bule de chá
422	Entidade improcessável (WebDAV)
423	Fechado (WebDAV)
424	Falha de Dependência (WebDAV)
425	Coleção não ordenada
426	Upgrade Obrigatório
450	Bloqueados pelo Controle dos Pais do Windows

499	Cliente fechou Pedido (utilizado em ERPs/VPSA)
Outros Erros	
500	Erro interno do servidor
501	Não implementado
502	Bad Gateway
503	Serviço indisponível
504	Gateway Time-Out
505	HTTP Version not supported

Fonte: (HTTP Status Codes, 2017)

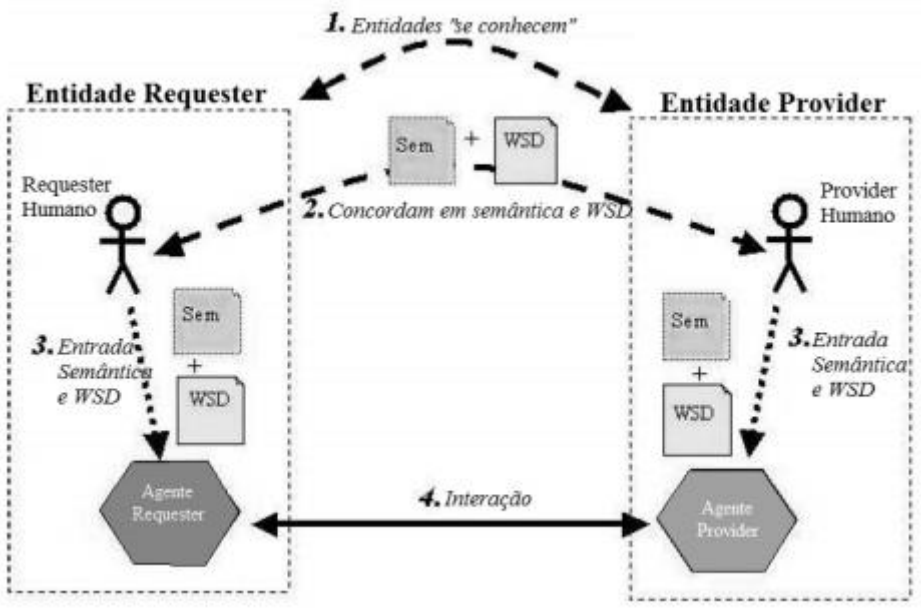
2.2 Arquitetura para Serviços Web

O computador vem se tornando uma ferramenta essencial e imprescindível para a realização de atividades de pessoas e instituições, desde as mais básicas até as complexas. Concomitante a essa crescente dependência dos sistemas computacionais, a complexidade dos softwares vem aumentando tão rapidamente que as técnicas para lidar com a complexidade de problemas existentes precisam evoluir. A Engenharia de Software lida com esses problemas, tornando então imprescindível, principalmente em sistemas de grande porte, projetar a estrutura global do sistema antes de iniciar seu desenvolvimento, ou seja, desenvolver o software orientado a uma arquitetura. A partir da necessidade de um projeto arquitetural para os sistemas que surgiu a Arquitetura de Software (Rodrigues, 2009).

No final do ano 2000 foi criado um grupo no W3C (Consórcio World Wide Web) com propósito de desenvolver uma arquitetura que permitisse a interoperabilidade entre aplicações e sistemas de plataformas, ambientes e arquiteturas diferentes. Visando atender a essas necessidades, foi proposta uma arquitetura computacional voltada para serviços Web, tornando possível a interação entre diferentes tipos de aplicações mesmo rodando em plataformas e sistemas operacionais diferentes (WSA, 2004).

A arquitetura de Serviços Web tem como base a referência arquitetural do W3C, o WSA (Web Service Architecture). A figura abaixo ilustra o processo geral de um Serviço Web (WSA, 2004):

Figura 1 - Processo Geral de um Serviço Web



Fonte: WSA, 2004

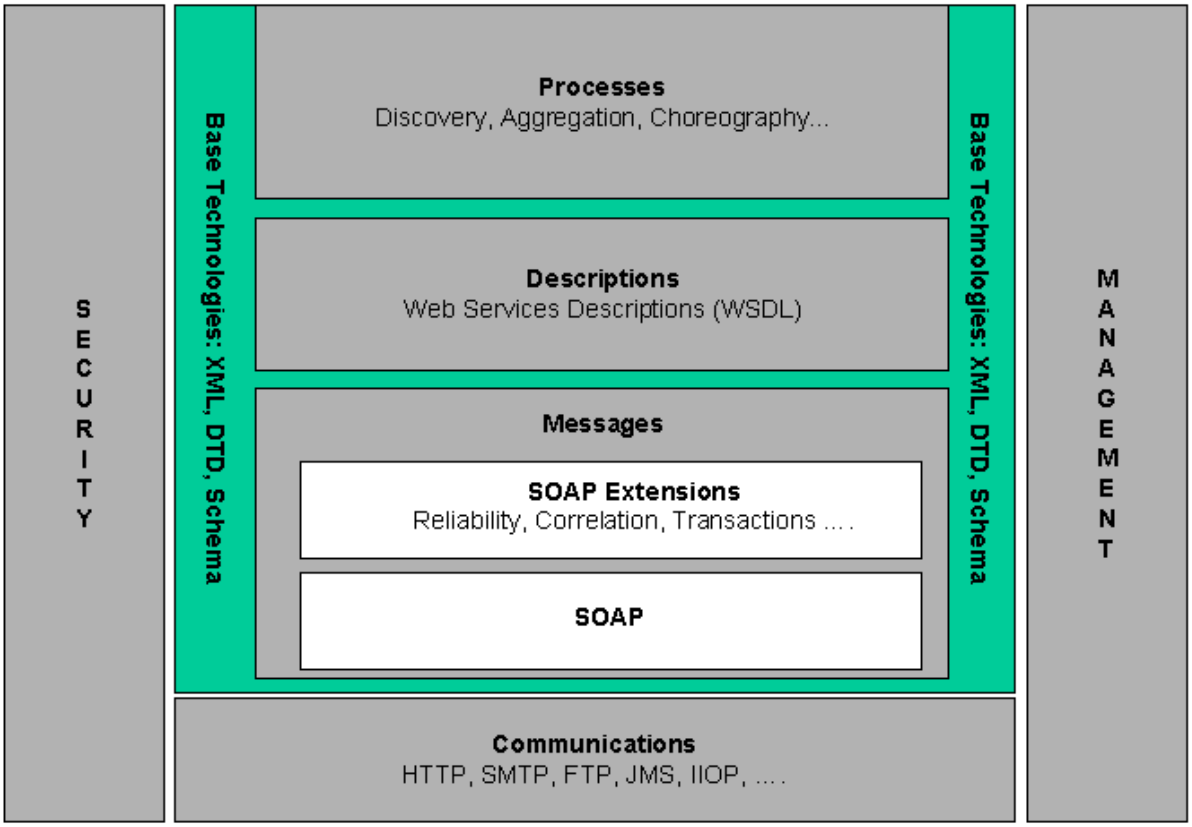
Há muitas maneiras pelas quais uma entidade pode acessar e usar um Serviço Web. Em geral, os passos necessários ilustrados na figura são:

- As entidades Requester e Provider são conhecidas.
- As entidades Requester e Provider chegam a um acordo em relação ao WSD (Web Services on Devices) e a semântica que irá coordenar a interação entre os agentes.
- O WSD e a semântica são consumados pelos agentes.
- As entidades Requester e Provider trocam mensagens, através de seus respectivos agentes.

2.2.1 Tecnologias de Serviços da Web

A arquitetura de serviços da Web envolve muitas tecnologias em camadas e inter-relacionadas. Há muitas maneiras de visualizar essas tecnologias, assim como há muitas maneiras de criar e usar serviços da Web. A Figura abaixo fornece uma ilustração de algumas dessas famílias de tecnologia (WSA, 2004).

Figura 2 - Arquitetura de Serviços Web



Fonte: WSA, 2004

2.2.1.1 Extensible Markup Language - XML

XML (Extensible Markup Language) é um formato de dados padrão, flexível e particularmente extensível, o XML reduz significativamente a responsabilidade de implantar as muitas tecnologias necessárias para garantir o sucesso dos webservices. XML foi desenvolvido pelo XML Working Group formado sob o panorama do World Wide Web Consortium (W3C) em 1996 (W3C, Extensible Markup Language (XML) 1.0 (Fifth Edition), 2008).

As metas de design para a criação do XML foram:

1. XML deve ser diretamente utilizável pela Internet.
2. XML deve suportar uma ampla variedade de aplicações.
3. XML deve ser compatível com SGML.
4. Deve ser fácil escrever programas que processam documentos XML.
5. O número de recursos opcionais no XML deve ser mantido no mínimo absoluto, idealmente zero.
6. Os documentos XML devem ser legíveis e razoavelmente claros.
7. O design do XML deve ser preparado rapidamente.
8. O design do XML deve ser formal e conciso.
9. Documentos XML devem ser fáceis de criar.

10. A intensidade na marcação XML é de importância mínima.

Cada documento XML tem uma estrutura lógica e física. Fisicamente, o documento é composto por uma série de unidades chamadas entidades. Uma entidade pode se referir a outras entidades para originar sua inclusão no documento. Logicamente um documento é iniciado por uma entidade "raiz", o documento deve ser composto de declarações, elementos, comentários, referências de caracteres e instruções de processamento, todas indicadas no documento por marcação explícita, tendo sua estruturas aninhadas corretamente (W3C, Extensible Markup Language (XML) 1.0 (Fifth Edition), 2008).

2.2.1.2 Simple Object Access Protocol - SOAP

O SOAP (Simple Object Access Protocol) é um protocolo que fornece uma estrutura padrão, para trocar informações estruturadas entre pares em um ambiente distribuído e descentralizado, também fornece um mecanismo conveniente para recursos de referência como os cabeçalhos. Isso permite que o SOAP seja usado em uma grande variedade de sistemas, desde sistemas de mensagens até RPC (Chamadas Remotas de Procedimento) (W3C, Simple Object Access Protocol (SOAP) 1.1, 2000).

O SOAP consiste em três partes:

• O envelope SOAP define uma estrutura geral para expressar o que está em uma mensagem; quem deve lidar com isso, e se é opcional ou obrigatório.

• As regras de codificação SOAP definem um mecanismo de serialização que pode ser usado para trocar instâncias de tipos de dados definidos pelo aplicativo.

• A representação SOAP RPC define uma convenção que pode ser usada para representar chamadas de procedimento e respostas.

2.2.1.3 Web Services Description Language - WSDL

O WSDL (Web Services Description Language) foi submetida ao W3C por Ariba, IBM e Microsoft em março de 2001, apresenta os serviços da Web fundamentado em XML, que utilizam mensagens trocadas entre o solicitante e os agentes do provedor. As mensagens em si são apresentadas abstratamente e depois ligadas a um protocolo de rede e um formato de mensagem concretos. Permitindo assim que processos executados em

sistemas operacionais diferentes se comuniquem usando XML, fazendo com que os clientes de serviços da Web invoquem e recebam respostas independentemente do idioma e das plataformas (Hirsch, Kemp, & Ilkka, 2007).

2.3 Semântica de Serviços da Web

Para que os programas de computador interajam entre si com sucesso, condições de semântica necessitam ser aplicadas. A Semantic Web Services Ontology (SWSO), semântica aplicada aos serviços Web influencia todo o desenvolvimento e implantação de uma aplicação (W3C, 2005). Diferenças superficiais na especificação de um serviço podem fazer com que um solicitante não o encontre, mesmo que ele seja o serviço desejado. Um enorme passo para resolver este tipo de problema é não ser tão superficial na hora de encontrar e invocar o serviço, e identificá-lo a partir de analogias semânticas, ou seja, pelas suas capacidades (Morisson, 2010):

- Deve haver uma conexão física entre eles, de modo que os dados de um processo possam alcançar outro
- Deve haver concordância sobre a *forma* como os dados são tratados.
- Os dois (ou mais) programas devem compartilhar o significado pretendido dos dados.
- Deve haver concordância quanto ao processamento implícito de mensagens trocadas entre os programas

2.3 Arquitetura REpresentational State Transfer (REST)

Esta seção apresenta a Arquitetura REpresentational State Transfer (REST), um estilo arquitetônico para sistemas distribuídos, a engenharia de software descreve os princípios que orientam o REST e as restrições para sustentar esses princípios. REST é um estilo híbrido derivado de vários dos estilos arquiteturais baseados em rede e combinado com restrições adicionais que definem uma interface de conexão uniforme. A estrutura de arquitetura de software é usada para definir os elementos arquiteturais de REST e examinar o processo de amostra, o conector e as exibições de dados de protótipos arquiteturas, para a troca de dados entre webservices são utilizados os formatos XML e JSON (Fielding R. T., Architectural Styles and the Design of Network-based Software Architectures, 2000).

REST (Representation State Transfer) é um estilo arquitetural híbrido para sistemas distribuídos derivado da combinação de alguns estilos arquiteturais (Fielding R.

T., Architectural Styles and the Design of Network-based Software Architectures, 2000). Habitualmente guiado pelo Modelo de Maturidade de Richardson (FOWLER, 2010), o REST é fundamentado no vocabulário de métodos HTTP (ex.: GET, POST, PUT) para representar as ações que podem ser executadas pelo servidor (WSA, 2004).

O REST distingue três classes de elementos arquitetônicos (Jakl, 2005):

- Elementos de dados: O aspecto principal do REST é o estado dos elementos de dados, os componentes se comunicam transferindo representações do estado atual ou do estado desejado dos elementos de dados, utilizando identificadores para distinguir entre os diversos recursos. Em um ambiente da Web, o identificador é um Uniform Resource Identifier (URI), conforme definido no Internet RFC 2396.

- Elementos de conexão (conectores): Gerenciamento da comunicação de rede para os componentes. Os conectores apresentam uma interface abstrata trazendo vantagens como uma melhor separação de interesses na implementação, o ocultamento, simplicidade e substituibilidade da implementação.

- Elementos de processamento (componentes): São identificados por sua função dentro de uma aplicação, podem ser:

    - O agente usuário usa um conector de cliente para iniciar uma solicitação e se torna o destinatário final da resposta.

    - O servidor de origem usa um conector de servidor para receber a solicitação e a fonte definitiva de representações de seus recursos. Cada servidor fornece uma interface genérica para seus serviços como uma hierarquia de recursos.

    - Componentes intermediários agem como cliente e servidor para encaminhar - com possível tradução - solicitações e respostas

2.4 Javascript Object Notation - JSON

JSON (Javascript Object Notation) é um formato de serialização de dados legível por humanos baseado em texto com especificação padronizada e parcialmente descritivo. O JSON foi criado por Douglas Crockford que tinha como objetivo prático representar informações de forma simplificada, leve e flexível, e que pudesse obter desempenho superior ao formato XML. Com o passar do tempo, JSON se tornou popular, tendo se adaptado muito bem em ambientes de aplicações distribuídas. O JSON foi construído com base em quatro tipos primitivos de dados e outros dois para composição. Cada tipo

possui seu respectivo correspondente na maior parte das linguagens de programação, embora possam ser identificados por nomes distintos (Droettboom, 2015).

Quadro 2 - Tipos de valores em JSON

Tipo	Valor de Exemplo
object	{"chave": "valor", "chave": "valor"}
list	"lista" : [{"prop1":"value1", "prop2":"value2"}, {“prop1":"value3", "prop2":"value4”}]
array	["primeiro", "segundo", "terceiro"]
number	1, -1, 2.9999
string	"Conjunto de caracteres"
boolean	true, false
null	null

Fonte: (ECMA-404, 2017)

Por meio da composição de objetos, listas e tipos primitivos, é possível atingir a representação das mais variadas estruturas de dados, sejam elas simples ou complexas. Dada uma estrutura, é possível que se represente de diversas formas. Abaixo (Figuras 3 e 4) estão duas formas distintas de realizar a representação de uma entidade pessoa através da linguagem JSON.

Figura 3 - Representação JSON de uma entidade pessoa

```
{
"nome": "Mateus da Silva",
"aniversario": "15 de maio de 1990",
"cidade": "Florianópolis, SC, Brasil"
}
```

Figura 4 - Representação alternativa em JSON de uma entidade pessoa

```
{
"nome": "Mateus",
"sobrenome": "da Silva",
"nascimento": "15-05-1990",
"cidade": {
"nome": "Florianópolis",
"estado": "SC",
"pais": "Brasil"
}
“telefones”: {
        “fixo”: ”(48) 3203-2760”
        “celular”: “(48) 98495-6532”
}
}
```

2.5 C#

C# é uma linguagem de programação criada por Anders Hejlsberg, uma lenda da programação que ingressou na Microsoft Corporation em 1996, após um período de 13 anos de carreira na Borland, onde foi o arquiteto-chefe de Delphi e Pascal. A linguagem C# foi criada junto com a arquitetura .NET. Embora existam várias outras linguagens que

suportam essa tecnologia (como VB.NET, C++, J#), C# é tida como a linguagem símbolo do .NET pois foi criada praticamente do zero para trabalhar na nova plataforma, sem preocupações de compatibilidade com código de legado, o compilador C# foi o primeiro a ser desenvolvido e maior parte das classes do .NET Framework foram desenvolvidas aproveitando C#. Após o renome da linguagem, a Microsoft submeteu-a à ECMA (European Computer Manufacturers Association), associação cujo objetivo é a padronização de sistemas de informação. Em 2001, a ECMA aprovou o C# e a linguagem recebeu a especificação ECMA-334. Mais tarde, em 2003, tornou-se padrão também da ISO, recebendo a especificação de ISO/IEC 23270 (ECMA-334, 2017).

C# é uma linguagem de programação simples, moderna, orientada a objetos e segura. Tem suas raízes na família C e, de maneira imediata se parece familiar a programadores de C, C++ e Java. A sintaxe de C# simplifica muitas complicações de C++ e fornece soluções poderosas como tipos de valor nulos, enumerações, expressões lambda e acesso direto à memória, que não existem em muitas linguagens. C# oferece suporte a tipos e métodos genéricos, o que proporciona mais segurança e desempenho para os tipos, e iteradores, que permitem aos implementadores definir os comportamentos personalizados da iteração simples de usar pelo código do cliente.

As expressões LINQ (Consulta Integrada à Linguagem) tornam a consulta fortemente tipada uma construção de linguagem de primeira classe. Por ser uma linguagem orientada a objeto, o C# oferece suporte aos conceitos de encapsulamento, herança e polimorfismo. Todas as variáveis e métodos, incluindo o método Main, o ponto de entrada do aplicativo, são encapsulados em definições de classe. Uma classe pode herdar diretamente de uma classe pai, e pode implementar qualquer quantidade de interfaces. Métodos que substituem métodos virtuais em uma classe pai exigem a palavra-chave override como uma forma de evitar uma redefinição acidental. Em C#, um struct é como uma classe simplificada; é um tipo alocado na pilha que pode implementar interfaces, mas não oferece suporte à herança. Além desses princípios básicos orientados a objeto, C# facilita o desenvolvimento de componentes de software por meio de várias construções de linguagem inovadoras, incluindo o seguinte (Liberty & MacDonald, 2010):

- Atributos, que fornecem metadados declarativos sobre os tipos no tempo de execução.
- Comentários embutidos da documentação XML.
- LINQ (Consulta Integrada à Linguagem) provê recursos de consulta internos em várias fontes de dados, inclusive acesso a bancos de dados formatando a consulta SQL com assistência do LINQ.

O processo de compilação de C# é simples comparado a C e C++, e mais flexível do que em Java. Não há arquivos de cabeçalho separado, e nenhum requisito de que os métodos e os tipos sejam declarados em uma ordem específica. Um arquivo de código-fonte de C# pode definir qualquer quantidade de classes, estruturas, interfaces e eventos. C # tem um sistema de tipo unificado. Todos os tipos C #, abrangendo tipos primitivos, como *int* e *double*, herda de um único tipo de objeto raiz. Assim, todos os tipos compartilham um conjunto de operações comuns e valores de qualquer tipo podem ser armazenados, transportados e operados de forma sólida. Além disso, o C # suporta tipos de referência e tipos de valores definidos pelo usuário, permitindo alocação dinâmica de objetos, bem como armazenamento em linha de estruturas leves.

Aspectos do design do C# que foram diretamente influenciados por considerações de versão incluindo os modificadores virtual e de substituição separados, as regras para a resolução de sobrecarga do método, e suporte para declarações explícitas de membros da interface (Liberty & MacDonald, 2010).

2.6 Model-View-Controller - MVC

Modelo de arquitetura de projetos que possui três camadas (Modelo, Visualização e Controle) e divide a aplicação de modo que a lógica de negócio seja a camada central dentre as três camadas físicas. Isto é chamado de camada física de negócios. Grande parte do código escrito deve estar na camada de apresentação e de negócio (Reenskaug & Coplien, 2009).

A arquitetura MVC possui uma maneira de dividir a funcionalidade envolvida na manutenção e apresentação dos dados de uma aplicação. Foi originalmente desenvolvida para mapear tarefas tradicionais de entrada, processamento e saída para o modelo de interação com o usuário. Este padrão facilita ao desenvolvedor o mapeamento de conceitos no domínio de aplicações Web multicamadas (Dooley, 2011).

O Modelo (Model) representa os dados da aplicação e regras de negócio que produzem acesso e modificação de informações. O modelo mantém o estado persistente do negócio e permite ao controlador (Controller) a habilidade de acessar funcionalidades da aplicação encapsuladas.

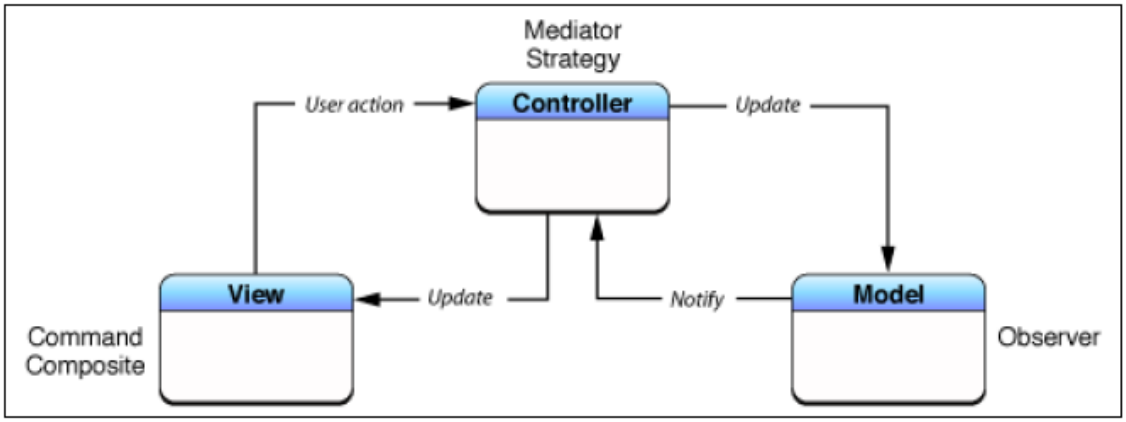
Um componente de Visualização (View) renderiza o conteúdo de uma determinada parte do Modelo e manda ao Controller ações executadas pelo usuário na

interface, além de realizar o acesso das informações da Model por meio do Controller definindo de que forma os dados recebidos devem ser apresentados.

A camada Controladora (Controller) determina o comportamento da aplicação, é ela que interpreta atividades do usuário e realiza o mapeamento para chamadas do modelo. Em aplicações Web essas ações de usuário poderiam ser cliques em botões, campos de checagem, preenchimento de campos ou seleções de menus em determinadas áreas da página. As ações realizadas pelo modelo abrangem ativar processos de negócio ou alterar o estado do modelo. Com base na ação do usuário e no resultado do processamento do modelo, o controlador seleciona uma visualização a ser exibida como parte da resposta à solicitação do usuário. Há normalmente um controlador para cada conjunto de funcionalidades relacionadas (Microsoft, 2014).

A arquitetura de 3 camadas que está representada abaixo é uma implementação do modelo MVC. O modelo MVC está preocupado em separar a informação de sua apresentação.

Figura 5 - Arquitetura de 3 camadas MVC



Fonte: (Campozano, 2018)

2.7 Javascript

A linguagem de programação JavaScript foi criada por Brendan Eich no período em que trabalhou na Netscape com o nome de mocha. LiveScript passou a ser o nome oficial da linguagem ao ser lançada na versão beta do navegador Netscape 2.0, no segundo semestre de 1995. Quando houve o lançamento do navegador Netscape versão 2.0B3,

teve novamente seu nome mudado em um anúncio junto à Sun Microsystens em dezembro de 1995 para JavaScript (ECMA-262, 2018).

No momento em que a Netscape passou a dar suporte para a tecnologia Java no seu navegador ocorreu uma mudança de nome de LiveScript para JavaScript. A escolha do nome deixou a comunidade de desenvolvimento confusa, pois passou a impressão de que a linguagem era baseada em Java. Na época, o nome escolhido foi compreendido de forma a ser estratégia de marketing para surfar na onda da então recém lançada linguagem de programação Java. JavaScript rapidamente contraiu grande aceitação como linguagem de script client-side de páginas web.

Com o passar do tempo se transformou na linguagem de programação de maior popularidade na web. No entanto, em seu nascimento houve grande desconfiança e muitos profissionais chegaram a infamar a linguagem. Com o surgimento do Ajax (Asynchronous JavaScript e XML), que representa o uso do objeto XMLHttpRequest para comunicação com scripts de servidor, que pode enviar e receber informações em diversos formatos (JSON, XML, HTML, arquivos de texto)), o JavaScript com o passar do tempo teve sua fama aumentada e passou a receber cada vez mais reconhecimento em ambientes profissionais como grande recurso para desenvolvedores no desenvolvimento de aplicações web. Como resultado desta popularidade, houve uma proliferação de bibliotecas e frameworks, práticas de programação melhoradas e o aumento no uso do JavaScript fora do ambiente de navegadores (Rauschmayer, 2014).

Essa linguagem é usada para escrever funções incluídas em páginas HTML que interagem com o DOM (Document Object Model) da página. Com JavaScript pode-se, por exemplo, abrir uma nova janela do navegador com controle sobre seu tamanho, posição e atributos, validar valores de um formulário, garantindo validação de dados antes do envio ao servidor, mudar imagens e ações do mouse sob a tela, dentre tantas outras.

Como o código é executado localmente no navegador do usuário, o navegador envia respostas às ações rapidamente, deixando a aplicação mais dinâmica. Além disso é possível detectar ações do usuário, como teclas pressionadas individualmente, movimentação do mouse, descanso do mouse, etc. Em função de JavaScript ser a única linguagem que a grande maioria dos navegadores mais populares suportam (Google Chrome, Mozilla Firefox, Internet Explorer, Edge, Safari, Opera), cada vez mais a linguagem se torna alvo de frameworks em outras linguagens (Rauschmayer, 2014).

2.8 SQL Server

A necessidade de armazenar e tratar os dados gerados em uma aplicação WEB existe para a grande maioria das aplicações atuais. Em função da aplicação desenvolvida neste projeto também tratar informações enviadas e recebidas de diversas aplicações ao redor do globo, a tecnologia utilizada para tratar e armazenar os dados a serem gerados na execução da aplicação foi SQL Server (Delaney, 2000).

O banco de dados SQL Server é um sistema gerenciador de banco de dados relacional (SGBD) desenvolvido pela Microsoft. Sua criação se deu em função de uma parceria com a Sybase em 1988 inicialmente para a plataforma OS/2.Esta parceria se manteve por durante 6 anos, quando em 1994 a Microsoft passou a manter sozinha a manutenção do gerenciador de banco de dados. No ano de 1995 a Microsoft lançou a versão 6.0 do SQL Server, que foi uma das maiores reescritas da tecnologia SQL Server. Esta versão 6.0 melhorou de forma bastante significativa a performance da aplicação, provendo mecanismos internos de replicação e administração centralizada. Em 2000 foi lançado o SQL Server 2000, considerado o mais importante do SQL Server até o momento. Tal versão foi construída sobre o framework do SQL Server 7.0. Estas mudanças foram desenvolvidas para tornar essa tecnologia mais nova pelos próximos 10 anos (Silberschatz, Sundarshan, & Kort, 2016).

Nesses anos todos, o SQL Server cresceu e foram criadas diversas versões diferentes que variam de aplicações de pequeno porte, para um ou alguns poucos computadores conectados, até aplicações com milhões de usuários espalhados em todo mundo com acesso simultâneo a todo instante, salvando, atualizando, excluindo e lendo quantidades gigantescas de dados por segundo. As linguagens de consulta primárias desta tecnologia são Transact-SQL (T-SQL) e ANSI SQL. Em termos de ferramenta para armazenamento e manipulação de dados, o SQL Server ao longo dos anos despontou como principal ferramenta da Microsoft e também um dos maiores e mais utilizados SGBDs relacionais do mercado.

O software permite a concepção de tabelas relacionadas, evitando a necessidade de armazenar dados redundantes em várias localidades dentro de um banco de dados. O modelo relacional também provê integridade referencial e diversas restrições de integridade para manter a precisão dos dados.

O SQL Server nada mais é do que um gerenciador de bases de dados relacionais, ou seja, as informações manipuladas com SQL Server são armazenadas em tabelas.

Adicionalmente, o sistema apresenta recursos avançados para promover a atualização dos dados e garantir que as informações guardadas se mantenham corretas e confiáveis. Entre as funcionalidades deste SGBD, pode-se destacar que o SQL Server atua com sistemas de criptografia integrada, o que garante com que os dados exclusivamente serão visualizados, removidos ou alterados por usuários que possuem autorização para tal. Além disso, o SQL Server permite que o administrador do sistema determine permissões para acesso apenas a determinadas tabelas ou então determinadas atividades, como, por exemplo, um usuário pode realizar a consulta dos dados, mas não remover e nem adicionar dados. Adicionalmente, oferece recursos de log, registros de acesso e operações realizadas em determinado período, o que permite realizar trilhas de auditoria para identificação de responsabilidades (Delaney, 2000).

Em relação à integridade, pode-se destacar que com o uso de controles sobre os dados, o SQL Server não permite que sejam geradas inconsistências que venham a inviabilizar a utilização precisa dos dados.

O banco de dados desenvolvido e mantido pela Microsoft também possibilita que mais de um usuário acesse a mesma tabela ao mesmo tempo. Para isso, existe um controle inteligente que mantém a consistência das informações, isto é, garantindo que os dados estejam devidamente atualizados. Tal garantia se dá pelo uso de bloqueios parciais, alertas e também da gravação de versões anteriores dos dados, como forma de consolidar a base de dados (Delaney, 2000).

2.9 Testes de Software

Com a adoção de serviços REST alinhados à estratégia de serviços Web, que cada vez mais se tornam amplamente difundidos pela comunidade de desenvolvedores, percebemos nos resultados do desenvolvimento a geração de diversos artefatos independentes. Em função de diversas APIs geradas constantemente, cada vez mais se faz necessário a realização de testes destas APIs para a detecção da ocorrência de erros no processo de construção do programa.

Os testes podem ser classificados de duas maneiras: teste baseado em especificação (specification-based testing) e teste baseado em programa (program-based testing). De acordo com tal classificação, têm-se que os critérios da técnica funcional são baseados em especificação e tanto os critérios estruturais quanto baseados em erros são considerados critérios baseados em implementação. Diferentes métodos de testes não

devem ser vistos como técnicas concorrentes, mas como métodos complementares usados para detectar diferentes tipos de erros. (Howden, 1987).

O **teste funcional** também é conhecido como teste caixa preta pelo fato de tratar o software como uma caixa cujo conteúdo é desconhecido e da qual só é possível visualizar o lado externo, ou seja, os dados de entrada fornecidos e as respostas produzidas como saída. Na técnica de teste funcional são verificadas as funções do sistema sem se preocupar com os detalhes de implementação (Myers G. J., 1979).

A **técnica estrutural** é um método de projeto de testes que usa a estrutura de controle do projeto procedimental para derivar casos de teste, apresentando uma série de limitações e desvantagens, decorrentes das limitações inerentes às atividades de teste de programa enquanto estratégia de validação, pois não é viável testar todos os caminhos lógicos de um programa (Rapps & Weyuker, 1985). Esses aspectos introduzem sérios problemas na automatização do processo de validação de software. Independentemente dessas desvantagens, essa técnica é vista como complementar à técnica funcional e informações obtidas pela aplicação desses critérios tem sido consideradas relevantes para as atividades de manutenção, depuração e confiabilidade de software (Pressman, 1997).

A técnica de **teste baseada em erros** utiliza informações sobre os tipos de erros que acontecem com mais frequentes no processo de desenvolvimento de software para derivar os requisitos de teste. A ênfase da técnica está nos erros que o programador ou projetista pode cometer durante o desenvolvimento e nas abordagens que podem ser usadas para detectar a sua ocorrência. Semeadura de Erros (Error Seeding) e Análise de Mutantes (Mutation Analysis) são critérios típicos que se concentram em erros. (DeMillo, Lipton, & Sayward, 1978).

Os testes de software são utilizados para o controle de qualidade, assegurando que o software esteja contemplando todas as funcionalidades esperadas e que as mesmas estejam funcionando corretamente, para otimizar seu trabalho, o testador ou equipe de testes, pode utilizar ferramentas de automatização. Com isso, há um ganho de tempo muito grande, pois a ferramenta pode realizar testes pré-gravados de forma automática e mais rápida.

O design de casos de teste de um sistema requer mais criatividade, inteligência e experiência do que as necessárias para projetar o sistema ou programa de acordo com Myers. Existem 17 tipos de testes que são exibidos no Quadro 3 abaixo. Não é necessário

que todos os 17 tipos sejam aplicadas a todos os programas, mas, para evitar negligenciar algo, todas os 17 tipos devem ser explorado ao projetar casos de teste (Myers G. J., 2004). Lista dos tipos de testes que podem ser feitos:

Quadro 3 - Tabela dos tipos de testes

Tipo de Teste	Descrição
Teste de Unidade	Teste em um nível de componente ou classe. É o teste cujo objetivo é um "pedaço do código".
Teste de Integração	Garante que um ou mais componentes combinados (ou unidades) funcionam. Podemos dizer que um teste de integração é composto por diversos testes de unidade.
Teste Operacional	Garante que a aplicação pode rodar muito tempo sem falhar.
Teste Positivo-negativo	Garante que a aplicação vai funcionar no "caminho feliz" de sua execução e vai funcionar no seu fluxo de exceção.
Teste de Regressão	Toda vez que algo for mudado, deve ser testada toda a aplicação novamente.
Teste de Caixa-preta	Testar todas as entradas e saídas desejadas. Não se está preocupado com o código, cada saída indesejada é visto como um erro.
Teste Caixa-branca	O objetivo é testar o código. Às vezes, existem partes do código que nunca foram testadas.
Teste Funcional	Testar as funcionalidades, requerimentos, regras de negócio presentes na documentação. Validar as funcionalidades descritas na documentação (pode acontecer de a documentação estar inválida)
Teste de Usabilidade	Verifica se a navegabilidade e os objetivos da tela funcionam como especificados e se atendem da melhor forma ao usuário.

Teste de Performance	Verifica se o tempo de resposta é o desejado para o momento de utilização da aplicação.
Teste de Carga	Verifica o funcionamento da aplicação com a utilização de uma quantidade grande de usuários simultâneos.
Teste de Aceitação do usuário	Testa se a solução será bem vista pelo usuário. Ex: caso exista um botão pequeno demais para executar uma função, isso deve ser criticado em fase de testes. (aqui, cabem quesitos fora da interface, também).
Teste de Volume	Testar a quantidade de dados envolvidos (pode ser pouca, normal, grande, ou além de grande).
Testes de Stress	Testar a aplicação sem situações inesperadas. Testar caminhos, às vezes, antes não previstos no desenvolvimento/documentação.
Testes de Configuração	Testar se a aplicação funciona corretamente em diferentes ambientes de hardware ou de software.
Testes de Instalação	Testar se a instalação da aplicação foi OK.
Testes de Segurança	Testar a segurança da aplicação das mais diversas formas. Utilizar os diversos papéis, perfis, permissões, para navegar no sistema.

Fonte: (Myers G. J., 2004)

Ao criar os cenários de testes para APIs REST, é necessário que se ofereça um amplo conhecimento sobre sua documentação, uma API bem documentada é de grande ajuda, pois através dela saberemos os recursos, corpo de mensagem presentes nas respostas.

Dados de uma requisição:

- URI
- Método
- Headers
- Parameters
- Body

Dados de uma resposta:

- Status Code
- Headers
- Body

Comportamento da requisição:

- Validar dados retornados.
- Validar header da resposta
- Validar se a resposta está conforme o esperado.
- Validar se, ao alterar o content-type, o comportamento se mantém.
- Validação da estrutura JSON ou XML.
- Validar ao gerar um erro se o status está correto.
- Validar o comportamento de uma requisição com informações incompletas.

2.9.1 Casos de Testes

Um caso de teste é uma especificação das entradas, condições de execução, procedimento de teste e resultados esperados que definem um único teste a ser executado para chegar a um objetivo específico de teste de software, como desempenhar um determinado caminho do programa corretamente ou verificar a conformidade com um determinado requerimento. Os casos de teste são importantes porque o teste completo é impossível, a estratégia óbvia, então, é tentar fazer testes o mais completo possível. Dadas as restrições de tempo e custo, é necessário definir um subconjunto dos possíveis casos de teste que tem a maior probabilidade de detectar a maioria dos erros (Myers G. J., 1979).

2.9.2 Automatização de Atividade de Teste

A qualidade e produtividade da atividade de teste são dependentes do critério de teste utilizado e da existência de uma ferramenta de teste que o suporte. Sem a existência de uma ferramenta automatizada a aplicação de um critério torna-se uma atividade propensa a erros e limitada a programas muito simples.

Outro fator importante é o suporte oferecido pelas ferramentas aos testes de regressão. Os casos de teste utilizados durante a atividade de teste podem ser facilmente obtidos para revalidação do software após uma modificação. Com isso, é possível checar se a funcionalidade do software foi alterada, reduzir o custo para gerar os testes de

regressão e comparar os resultados obtidos nos testes de regressão com os resultados do teste original (Souza, 1996).

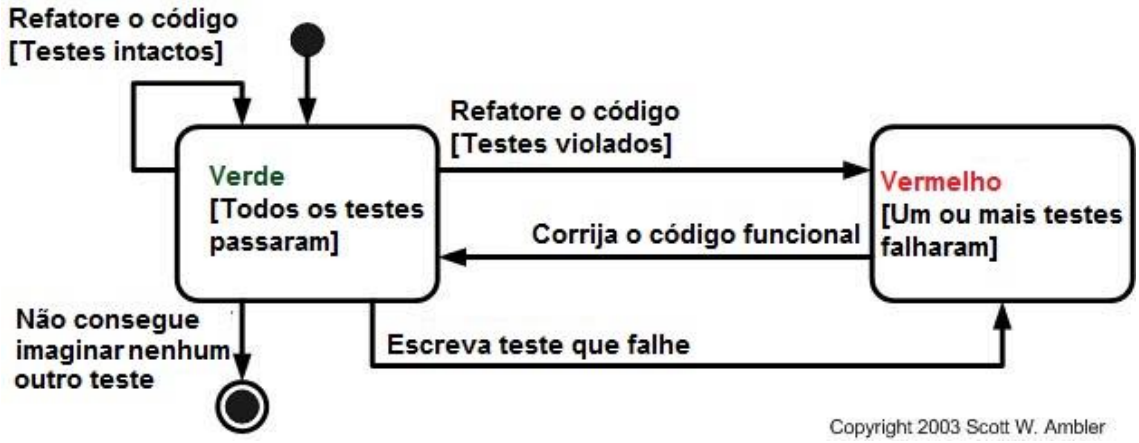
2.9.3 Test-Driven Development - TDD

O TDD tem suas raízes nos modelos de processos iterativos, incrementais e evolutivos antes de 1950. É acrescentado que o projeto da NASA Mercúrio de 1950 aplicou TDD, que não era conhecido até 2003, quando Kent Beck revelou este método (Saiedian & Janzen, 2005). TDD é uma técnica de desenvolvimento de software que se dá pela repetição disciplinada de um ciclo curto de passos de implementação de testes e do sistema (Koskela, 2007).

O ciclo de TDD é definido pelos seguintes passos:

1. Implementar um caso de teste;
2. Implementar um trecho do código suficiente para o novo caso de teste ter sucesso de tal modo que não quebre os testes previamente escritos;
3. Se necessário, refatorar o código produzido para que ele fique mais organizado;

Figura 6 - Ciclo de atividades TDD



Fonte: Ambler, 2003

A capacidade do TDD de melhorar a qualidade, simplicidade e capacidade de teste, enquanto aumenta a confiança no código e a produtividade da equipe, quando comparado ao modelo tradicional (testes realizados no final do projeto). O resultado obtido através do seu levantamento é que o TDD possui módulos menores com menos métodos e maior cobertura no código (ou seja, o número de linhas de codificação que são

executadas em casos de teste). Assim, é possível supor que, usando módulos menores, os autores atingiram ou superaram suas expectativas em termos de requisitos, em comparação com a abordagem tradicional. Um resultado direto é menos complexidade no desenvolvimento, o que resulta em manutenção mais fácil (Saiedian & Janzen, 2005).

3 SOLUÇÕES EXISTENTES

Para o desenvolvimento deste trabalho, foram pesquisadas as ferramentas existentes no mercado para o mesmo software proposto nesta monografia. Abaixo seguem alguns encontrados:

3.1 PAW

O Paw é uma ferramenta HTTP que permite testar e descrever APIs REST. Tem interface nativa MacOS para compor requisições, inspecionar respostas do servidor, gerar códigos de clientes e exportar definições de API. O PAW é uma ferramenta paga, propõe 30 dias de teste no plano gratuito e após isso custa $49,99 dólares. O Paw é exclusivamente de uso para computadores com sistema operacional MacOS (Paw, 2018).

3.2 SoupUI

SoapUI é uma ferramenta de teste funcional que possui uma versão gratuita e uma paga. A versão gratuita é de código aberto e consente que se tenha acesso ao código-fonte, permitindo que seja modificado e customizado logo que seja necessário. A versão paga é mais amigável ao usuário e tem funcionalidades adicionais, incluindo um editor de formulários, um ajudante de asserção para XPath e um construtor de consultas SQL. Esta ferramenta é desktop, não existe solução para uso na web (Smartbear, 2018).

3.3 RestSharp

Ferramenta simples para realização de requisições REST para .NET. Esta ferramenta se limita ao uso no desenvolvimento de aplicações Dotnet. É uma ferramenta gratuita e pode ser obtida via GitHub ou gerenciador de pacotes Nuget da Microsoft. Esta ferramenta pode ser utilizada em .NET 3.5 ou superior, Silverlight 5, Windows Phone 8 ou Android. Oferece suporte a GET, POST, PUT, PATCH, HEAD, OPTIONS, DELETE (RestSharp, 2018).

3.4 RoboHydra

O RoboHydra é um servidor da Web projetado para testes HTTP, HTTPS ou WebSockets. Realiza testes em aplicativos móveis que se comunicam com algum servidor front-end desenvolvido em Javascript que utilizam servidores como repositórios de dados, aplicativos de servidores que requisitam servidores de terceiros. Para uso do Robohydra, é necessário que, além da instalação do software em si, também se instale o Node, é uma ferramenta gratuita e possui seu código no Github (RoboHydra, 2018).

3.5 Citrus Integration Testing

Esta ferramenta realiza testes HTTP REST, JMS, TCP/IP, SOAP, FTP, SSH, XML, JSON dentre outros, no entanto, é voltada para uso em automatização de testes, funcionando também como um framework de implementação para diversos tipos de testes voltados para uso de desenvolvedores de software e desenvolvedores de testes (ConSol, 2018).

3.6 POSTMAN

O Postman é um cliente para realizar testes e documentar APIs. Criado em 2012 o Postman aprimorou-se para apresentar suporte a todos os recursos necessários para o desenvolvimento e teste de API. É possível realizar solicitações HTTP como GET, POST, PUT e DELETE. É possível também executar o teste de automação em uma API. Existem uma solução de monitoramento, onde pode ser executado um conjunto de solicitações para verificar seu desempenho e resposta. Existe uma versão do aplicativo que contém recursos avançados, como suporte ao OAuth 2.0 além de upload e importação em massa. A ferramenta está disponível como uma extensão do Chrome e uma aplicação nativa para MAC, Windows e Linux (Postman Learning Center, 2018).

Em sua versão gratuita, o Postman propõe uma ferramenta de testes bastante completa, sendo considerada atualmente a de maior aceitação no mercado. Para sua versão Pro, os recursos se estendem a:

- Colaboração de equipe para API ou para revisão de históricos e atualizações recentes.

- Documentação da API para compartilhar documentação via web.
- Servidores para simular a API real e desacoplar equipes.
- Monitoramento de coleções para verificar o desempenho, o tempo de atividade e a exatidão de sua API.
- Integrações com outras ferramentas a partir da API do Postman.

A ferramenta desenvolvida tem como principal diferencial ser uma forma simples de cadastro de diversos testes de requisições REST

3.7 TRABALHOS CORRELATOS

Neste capítulo são identificados os trabalhos relacionados a partir de uma revisão sistemática da literatura utilizando termos de busca relacionados a este trabalho.

3.7.1 Planejamento da revisão

De modo a levantar quais seriam os trabalhos relacionados, foram selecionadas duas diferentes renomadas bases de dados bibliográficas para a execução da busca:
- IEEEXplore (http://ieeexplore.ieee.org).
- ACM Digital Library (http://portal.acm.org)

Foram definidos termos de busca relacionados aos objetivos deste trabalho, de modo a encontrar os resultados de maior relevância quando conectados os diferentes termos por AND:
- Termo 1: "test" or “testing”.
- Termo 2: "web application" or “web” or “web services”.
- Termo 3: "rest" or "Representational State Transfer".

Os trabalhos selecionados são mostrados a seguir:

Quadro 4 - Trabalhos Correlatos

Trabalho	Base
Test-the-REST: An Approach to Testing RESTful Web-Services (https://ieeexplore.ieee.org/document/5359602/?part=1)	IEEE
Metamorphic testing of RESTful web APIs (https://dl.acm.org/citation.cfm?id=3182528)	ACM
Connectedness testing of RESTful web-services (https://dl.acm.org/citation.cfm?id=1730902)	ACM

3.7.2 Relações com este trabalho

Nesta seção é realizada a análise dos trabalhos selecionados e comentado suas similaridades com este trabalho.

3.7.2.1 Test-the-REST: An Approach to Testing RESTful Web-Services

Chakrabarti e Kumar discorrem sobre uma abordagem para testes de serviços Web RESTful. Os autores também explicam que o projeto foi desenvolvido com uma série de inovações técnicas, com uma arquitetura baseada em plug-ins escaláveis, um formato de especificação de teste baseado em XML extensível e um método para reutilizar e compor casos de testes para casos de uso. Um protótipo de do projeto foi usado para testar um serviço RESTful.

3.7.2.2 Metamorphic testing of RESTful web APIs

Segura, Parejo, Troya e Ruiz-Cortés explicam como as APIs Web estão progressivamente se tornando a base da integração de software, e como sua validação está ficando cada vez mais crítica. Dentro deste contexto, a rápida detecção de bugs é de extrema importância para aumentar a qualidade de produtos internos e aplicativos de terceiros. Os autores fornecem uma alternativa quando a saída esperada de uma execução de teste é complexa ou desconhecido. Em vez de verificar uma única saída, o teste verifica se múltiplas execuções do programa cumprem certas propriedades.

3.7.2.3 Connectedness testing of RESTful web-services

Chakrabarti e Rodriquez discorrem sobre um algoritmo para testar a conectividade de web services RESTful, utilizando uma especificação formal do serviço da web. O algoritmo comentado pelos autores foi empregado para realizar testes automatizados do webservice, onde muitos defeitos funcionais além daqueles relacionados à conectividade foram detectados

4 DESENVOLVIMENTO

4.1 Solução Proposta

4.1.1 Levantamento de requisitos

Esta atividade tem como objetivo, compreender o problema, dando aos desenvolvedores, uma visão do que deve ser construído para resolução do problema. Desenvolvedores e clientes, em conjunto, tentam levantar e definir as necessidades do futuro usuário do sistema a ser implementado, tais necessidades são normalmente chamadas de requisitos (Bezerra, 2015).

Os requisitos de um sistema podem ser classificados como funcionais, não funcionais ou requisitos de domínio (Sommerville, 2011):

- Requisitos funcionais: são declarações de funções fornecidas pelo sistema. Descrevem a reação do sistema dadas determinadas entradas específicas e também como o sistema deverá se comportar sob as mais diversas situações. Em alguns casos, os requisitos funcionais podem descrever o que o sistema não deve fazer;
- Requisitos não funcionais: são restrições que agem sobre as funcionalidades providas pelo sistema. Como exemplo tem-se o tempo de execução de determinada tarefa, o espaço que o sistema deverá ocupar na memória e até mesmo características mais abstratas, como facilidade de utilização;
- Requisitos de domínio: têm sua origem em características do domínio da aplicação e não na especificação feita pelo cliente. Podem ser tanto funcionais como não funcionais. Este tipo de requisito pode parecer óbvio a um especialista, porém um desenvolvedor de software pode não o compreender, sendo assim a especificação tem de ser suficientemente compreensível.

4.1.2 Tecnologias utilizadas

O projeto foi desenvolvido utilizando a linguagem C#. Entende-se que esta linguagem atende completamente os requisitos, por ser uma das mais usadas no mundo para a programação Web. De fácil integração com os webservices e atendendo a necessidade por uma manutenibilidade mais acessível.

O banco de dados utilizado foi o SQLserver. Esse banco de dados foi escolhido porque apresenta boa performance e pode ser utilizado gratuitamente, preenchendo os requisitos necessários para o projeto.

4.1.3 Requisitos Funcionais

Segue lista dos requisitos funcionais deste sistema:

1. Cadastro de usuário. Informações referentes ao usuário do sistema, com os atributos nome, e-mail, curso, login e senha.
2. Autenticação. O sistema deve ter como tela inicial espaço para o usuário inserir login e senha. Também deve estar presente a opção para cadastro de novo usuário.
3. Cadastro de teste. Principal funcionalidade do sistema. Conterá informações para execução dos testes e a validação dos dados obtidos.
4. Lista de testes. O sistema deve exibir uma lista com os testes já cadastrados, permitindo sua edição e exclusão.
5. Verificação de resultados. O sistema deve exibir as informações referentes às avaliações obtidas a partir da execução de um teste.

4.1.4 Requisitos não funcionais

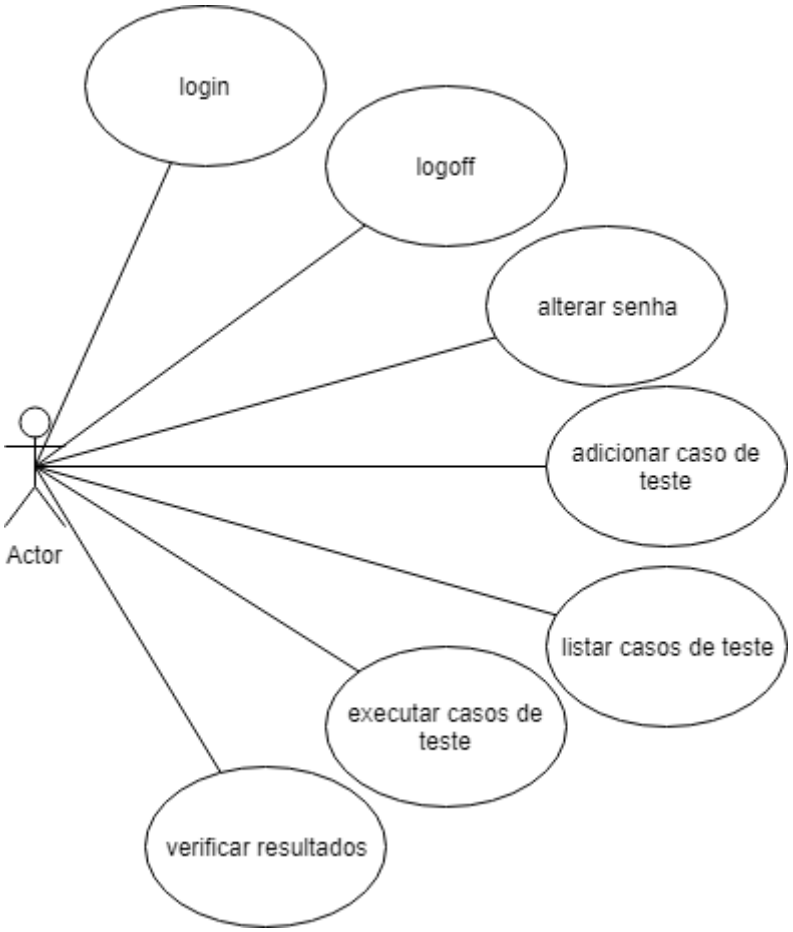
O sistema, por se tratar de um sistema web deverá rodar nos navegadores elencados neste requisito. O comportamento deve ser o mesmo, no que se refere às funcionalidades. Segue lista dos requisitos não funcionais deste sistema:

1. Compatibilidade. O sistema deve ser compatível com os navegadores Chrome, Firefox, Edge e Safari.

4.1.5 Diagrama de casos de uso

O diagrama de casos de uso documenta o que o novo sistema faz do ponto de vista dos usuários descrevendo as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema (G. Booch, 2005). Não foram aprofundados detalhes técnicos que dizem como o sistema faz. Apenas foram descritas as principais funcionalidades do sistema. Abaixo podemos compreender melhor como são os casos de uso:

Figura 7 - Casos de uso



4.1.5 Protótipo de interface de usuário

Com um protótipo é possível de forma mais rápida e econômica definir e experimentar um projeto, onde deverão estar dispostas na interface as informações da aplicação. Apenas durante a interação real do usuário com o sistema que os detalhes realmente são percebidos (PRESSMAN, 2005).

Figura 8 - Protótipo de baixa fidelidade da interface para login de usuários do sistema.

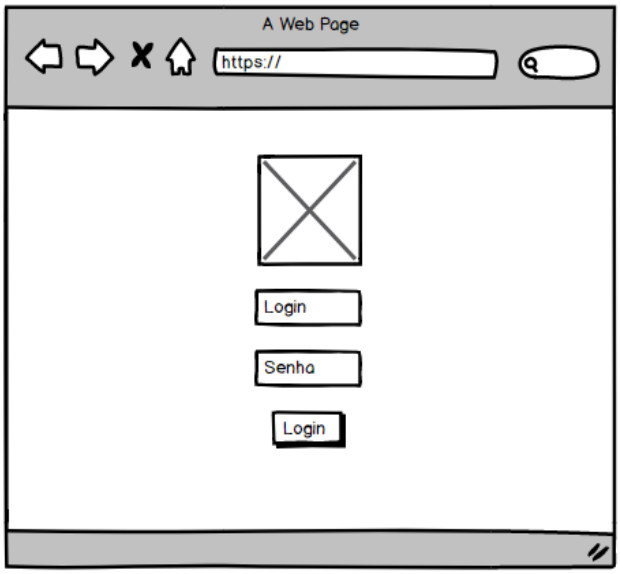


Figura 9 - Protótipo de baixa fidelidade da interface para tela de cadastro de teste

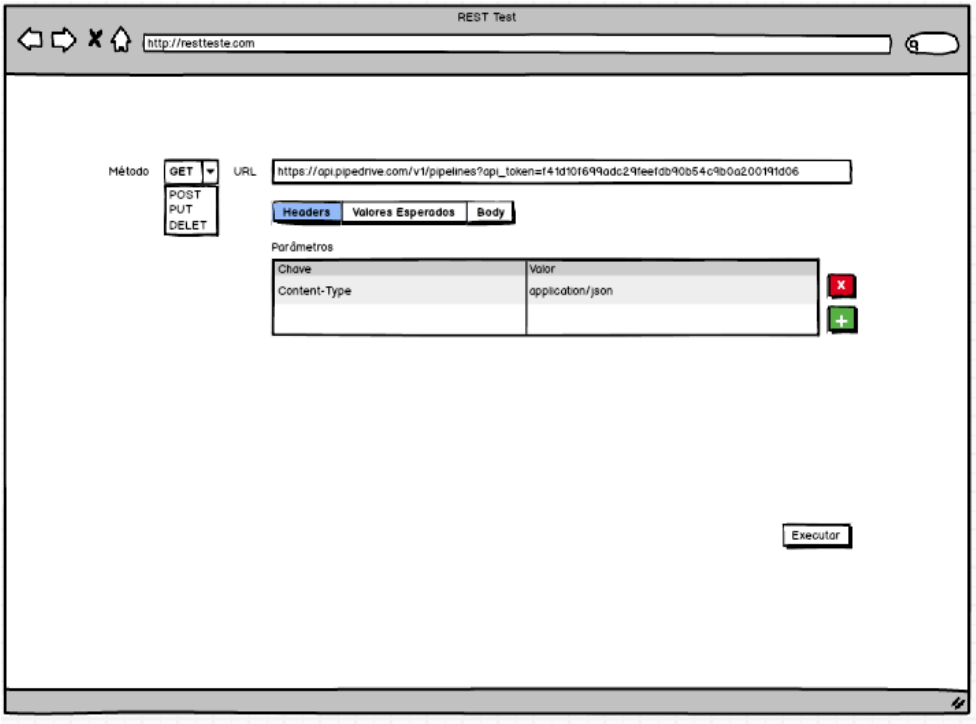
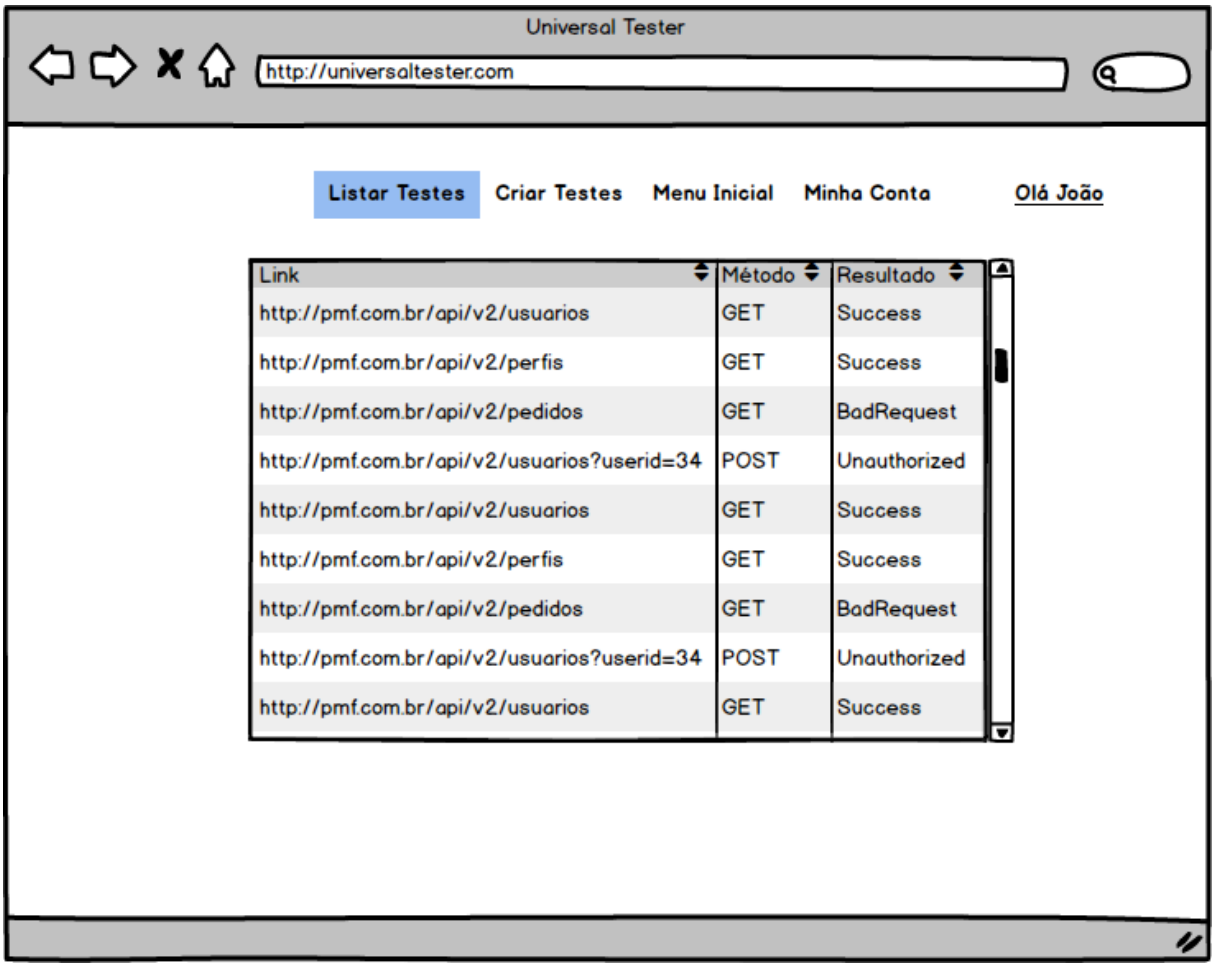
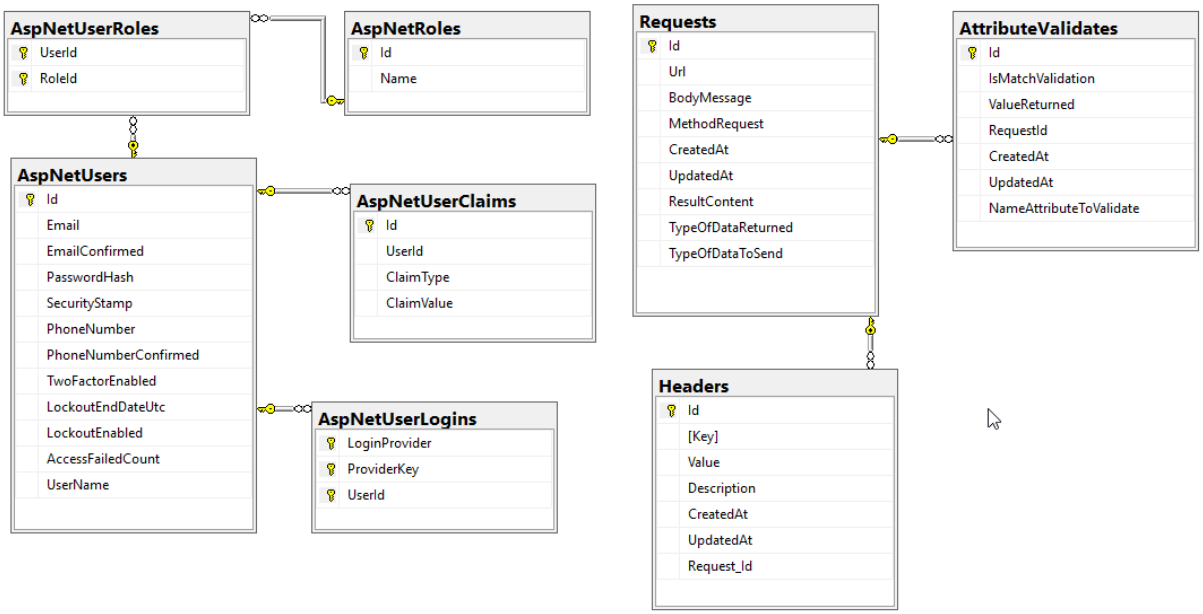


Figura 10 - Protótipo de baixa fidelidade da interface para tela de listagem de testes cadastrados



4.1.6 Modelagem do banco de dados

Figura 11 - Modelagem do banco de dados



Cada tabela contém informações necessárias para cada módulo do sistema. A saber:

- AspNetUserRoles;
- AspNetUsers;
- AspNetRoles;
- AspNetUserClaims;
- AspNetUserLogins;
- Requests;
- AttributeValidates;
- Headers.

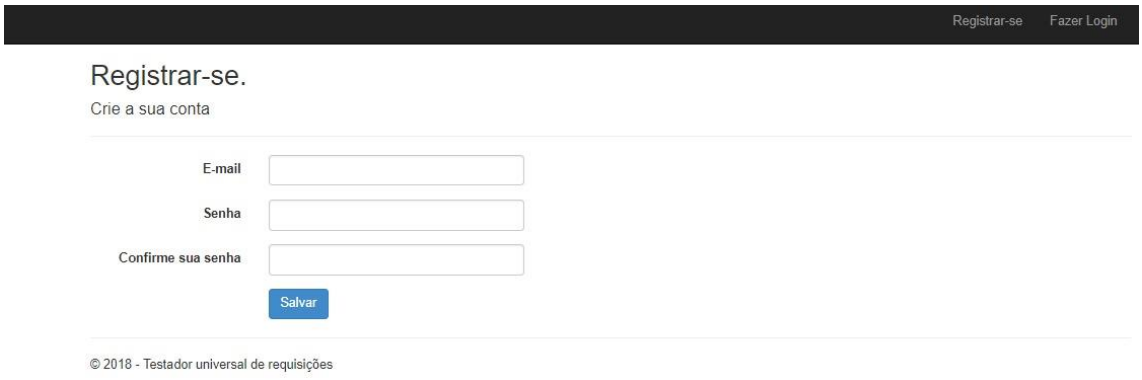
4.1.7 Funcionamento do software

No primeiro acesso ao sistema é exibido a tela de login, junto com formas de acessar a tela de registro de usuário por meio do link abaixo dos campos de login ou no menu superior.

Figura 12 - Tela de login

Na seção é onde o usuário faz o cadastro existem campos para serem inseridos um e-mail, senha e uma confirmação de senha. Após o cadastro o usuário obtém acesso ao sistema, podendo após o login, visualizar a tela de principal do sistema.

Figura 13 - Tela de Registro de usuário

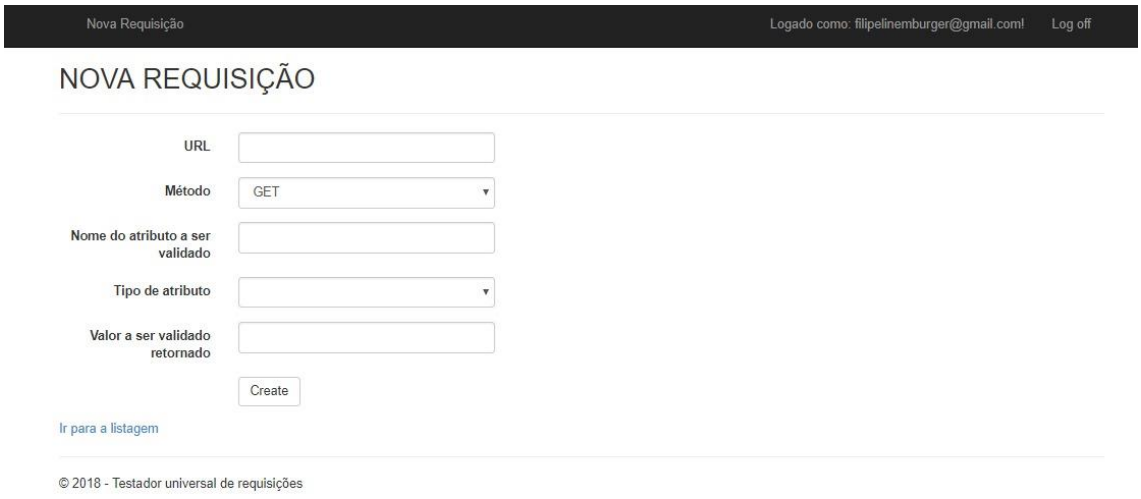


Na tela principal estão informações ao usuário que está utilizando o sistema e também informações referentes ao sistema. Nesta tela também existem opções para acessar a tela de cadastro de novo teste, ou diretamente a lista com os testes cadastrados.

Figura 14 - Tela principal

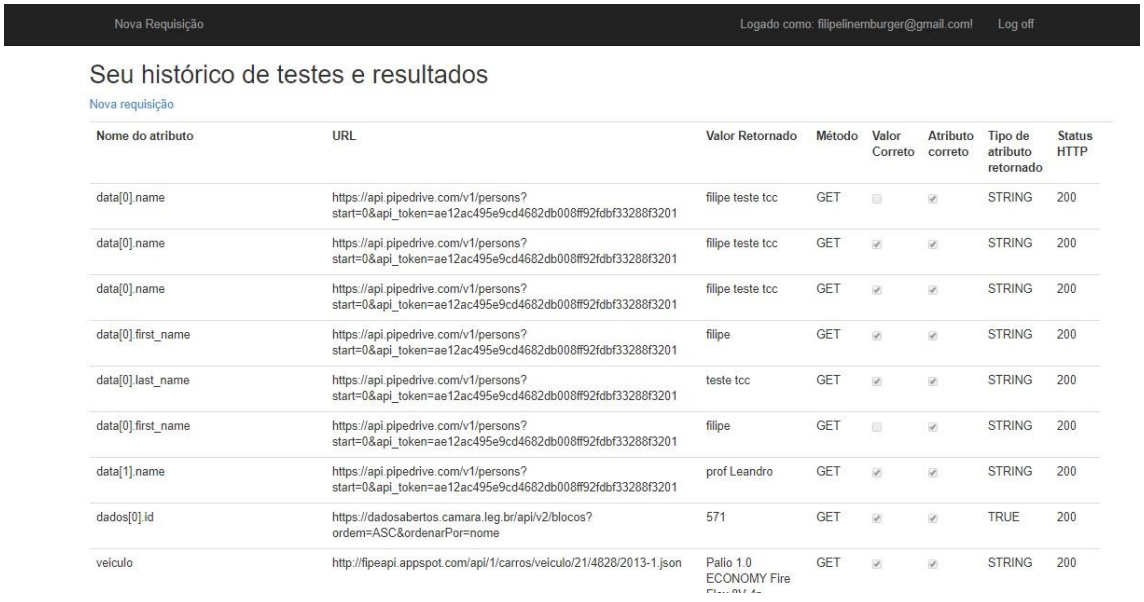
Na tela de cadastro de novo teste de requisição é necessário inserir a URL da requisição, o método HTTP utilizado no teste, o nome do atributo a ser validado dentro da resposta em formato JSON, o tipo e valor do atributo a ser validado, para assim poder ser validado e assumir que o teste criado está correto.

Figura 15 - Tela de cadastro de novo teste de requisição



Após a criação de algum teste, é possível acessá-lo pela lista de testes, onde são exibidas as informações cadastradas, contendo também informações sobre o resultado do teste após ser executado, exibindo se o valor ou atributo estão corretos e o status HTTP da resposta. Nesta tela também é possível executar novamente todos os testes cadastrados.

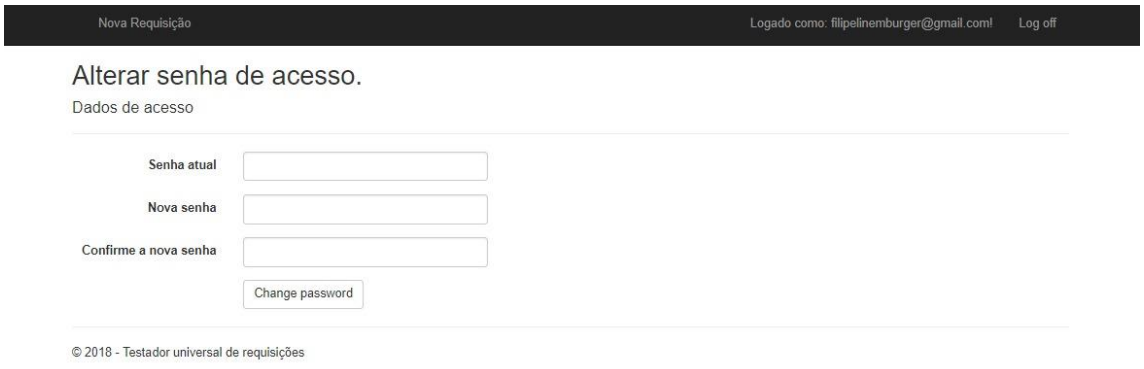
Figura 16 - Tela com lista de testes e resultados



Nome do atributo	URL	Valor Retornado	Método	Valor Correto	Atributo correto	Tipo de atributo retornado	Status HTTP
data[0].name	https://api.pipedrive.com/v1/persons?start=0&api_token=ae12ac495e9cd4682db008ff92fdbf33288f3201	filipe teste tcc	GET	☐	☑	STRING	200
data[0].name	https://api.pipedrive.com/v1/persons?start=0&api_token=ae12ac495e9cd4682db008ff92fdbf33288f3201	filipe teste tcc	GET	☑	☑	STRING	200
data[0].name	https://api.pipedrive.com/v1/persons?start=0&api_token=ae12ac495e9cd4682db008ff92fdbf33288f3201	filipe teste tcc	GET	☑	☑	STRING	200
data[0].first_name	https://api.pipedrive.com/v1/persons?start=0&api_token=ae12ac495e9cd4682db008ff92fdbf33288f3201	filipe	GET	☑	☑	STRING	200
data[0].last_name	https://api.pipedrive.com/v1/persons?start=0&api_token=ae12ac495e9cd4682db008ff92fdbf33288f3201	teste tcc	GET	☑	☑	STRING	200
data[0].first_name	https://api.pipedrive.com/v1/persons?start=0&api_token=ae12ac495e9cd4682db008ff92fdbf33288f3201	filipe	GET	☐	☑	STRING	200
data[1].name	https://api.pipedrive.com/v1/persons?start=0&api_token=ae12ac495e9cd4682db008ff92fdbf33288f3201	prof Leandro	GET	☑	☑	STRING	200
dados[0].id	https://dadosabertos.camara.leg.br/api/v2/blocos?ordem=ASC&ordenarPor=nome	571	GET	☑	☑	TRUE	200
veiculo	http://fipeapi.appspot.com/api/1/carros/veiculo/21/4828/2013-1.json	Palio 1.0 ECONOMY Fire	GET	☑	☑	STRING	200

Adicionalmente ao sistema, existe a tela de gerenciamento de senha de usuário, onde é possível alterar a senha já cadastrada.

Figura 17 - Tela para alterar senha de acesso


Nova Requisição
Logado como: filipelinemburger@gmail.com!
Log off
Alterar senha de acesso.
Dados de acesso
Senha atual
Nova senha
Confirme a nova senha
Change password
© 2018 - Testador universal de requisições

5 RESULTADOS OBTIDOS

Este trabalho resultou na criação de um sistema Web eficiente para realização de testes para requisições REST, proporcionando uma maior rapidez no teste destas requisições e possibilitando a avaliação dos mesmos.

Como base de comparação, o trabalho desenvolvido neste TCC tem como grande destaque o mapeamento de campos através da linguagem especificada de um determinado campo dentro de uma estrutura JSON Object. Dado que os softwares pesquisados que possuem finalidades semelhantes, não foi identificada esta mesma funcionalidade.

A especificação do teste implica que o usuário indique o nome do atributo que deve ser testado e o valor do atributo que deve ser retornado, podendo ser um retorno do tipo null.

Todos os web services testados têm em comum que a resposta é um objeto JSON, embora do ponto de vista conceitual, não seja errado que uma API retorne um tipo primitivo (boolean, string, number). Então o software analisa e válida apenas respostas no formato JSON, conforme foi encontrado no cenário de testes.

No levantamento feito, em algumas das APIs conhecidas que utilizam objeto JSON em suas respostas.

- Amazon
- Google
- LinkedIn
- Instagram
- GitHub
- Twitter
- https://developers.pipedrive.com/docs/api/v1/
- https://dadosabertos.camara.leg.br/swagger/api.html#api
- https://api.nasa.gov
- http://fipeapi.appspot.com/api/1/carros/veiculo/21/4828/2013-1.json
- http://www.triunfo.pe.gov.br/portal-transparencia/api/despesas
- https://api.pipedrive.com/v1/persons?start=0&api_token=ae12ac495e9cd4682db008ff92fdbf33288f3201

O sistema foi desenvolvido para dividir as requisições, não sendo necessário ocupar um grande espaço da memória. Suportando assim uma grande carga de requisições executadas ao mesmo tempo.

5.1 Avaliação

Para fins de métricas de teste, foram realizados testes com uma única, e também com dez requisições em diferentes API de diferentes portais como Google, Amazon e aplicações de CRM, e constatou-se que em média comparando a ferramenta Postman com o a ferramenta desenvolvida neste trabalho, o tempo de espera em milissegundos para as execuções está de acordo o quadro abaixo:

Quadro 5 - Métricas

Ação	**Ferramenta desenvolvida**	**Ferramenta PostMan**
Teste de uma (1) requisição	239 ms	348 ms
Teste de dez (10) requisições	2296 ms	2947 ms
Criar teste	7 segundos	5 segundos

Observando o resultado da avaliação, é visto que a ferramenta desenvolvida obtém dados de tempo próximos a ferramentas já disponíveis no mercado. Os valores obtidos na comparação entre a ferramenta desenvolvida e a ferramenta Postman não podem ser interpretados literalmente, pois os dados são médias de contagens realizadas em teste, e podem variar dependendo do momento e situação dos envolvidos, como a própria ferramenta, o servidor e principalmente a rede entre cliente e servidor.

6 CONCLUSÃO

Com este trabalho foi desenvolvido uma ferramenta eficiente para realização de testes para requisições REST, proporcionando uma maior rapidez no teste destas requisições, uma vez que é possível cadastrar diversos testes e executá-los quantas vezes quiser apenas com um clique.

A ferramenta desenvolvida implica que exista uma definição do teste, que deve ser cadastrado na ferramenta, explicitando o tipo da resposta e os valores que devem ser retornados na resposta da requisição. Durante a execução dos testes são validados o tipo da resposta da requisição e o valor específico do atributo da resposta.

Com a finalização do desenvolvimento foi averiguado a melhoria no tempo de execução de testes dado que agora todos os testes são cadastrados apenas uma vez e executados diversas vezes.

Acreditamos que a ferramenta desenvolvida será de grande ajuda no caso de softwares desenvolvidos utilizando o processo de desenvolvimento orientado por testes, ajudando na verificação e validação durante os ciclos de repetição de testes em aplicações desenvolvidas utilizando a técnica de TDD.

Observou-se a dificuldade de planejar e arquitetar a estrutura para a aplicação. Os objetivos específicos deste trabalho foram concluídos após a criação de uma aplicação Web objetiva e simplificada que permite a realização de testes em APIs com requisições REST.

Como observado em relação aos trabalhos correlatos, Chakrabarti e Kumar realizam abordagem para testes de serviços Web RESTful utilizando XML e desenvolvido um protótipo do projeto testado em um serviço RESTful. Comparando ao trabalho feito, a principal diferença é em relação a utilização de JSON. Segura, Parejo, Troya e Ruiz-Cortés fornecem uma alternativa quando a saída esperada de uma execução de teste é complexa ou desconhecido. Em vez de verificar uma única saída, o teste verifica se múltiplas execuções do programa cumprem certas propriedades. Comparado com a ferramenta desenvolvida, que pode apenas validar uma única saída. Chakrabarti e Rodriquez fornece um algoritmo empregado para realizar testes automatizados de um web servisse. A ferramenta desenvolvida utiliza um algoritmo diferente para realizar um teste automatizado em um web servisse.

6.1 Trabalhos Futuros

Com a finalização deste TCC, fica como ideia para continuação de desenvolvimento em trabalhos futuros a implementação deste mesmo serviço de testes com a utilização de XML no formato de resposta de uma requisição. Também é viável a implementação para utilização de autenticação no serviço a ser solicitado pela aplicação. É possível também adicionar novas formas de validação, como por exemplo, a validação de arrays e suas diversas variações.

Um serviço Web pode retornar dados de diversos tipos (string, number, boolean, object JSON). Embora seja possível que um webservice retorne esses tipos de dados, foi optado por ser possível apenas validar objetos JSON, por que durante a análise de diversos serviços Web, notamos que na prática, a grande maioria utiliza dados em formato object JSON como resposta para suas requisições, sendo incomum que um web service retorne apenas um boolean ou string, sem estar dentro de um objeto JSON.

Julgamos importante também criar uma forma de organizar testes em cenários, criando listas de testes diversas e gerenciadas individualmente, contribuindo assim, na melhora no planejamento, criação e execução dos testes.

REFERÊNCIAS

Benito-Osorio, D., Peris-Ortiz, M., Armengot, C. R., & Colino, A. (10 de 03 de 2018). *Web 5.0: the future of emotional competences in higher education.* Fonte: https://link.springer.com/article/10.1007/s40196-013-0016-5

Bezerra, E. (2015). *Princípios de análise e projeto de sistema com UML.*

Campozano, N. N. (20 de 11 de 2018). Fonte: As Camadas MVC: http://fabrica.ms.senac.br/2013/06/as-camadas-mvc/

ConSol. (10 de 07 de 2018). *Citrus Framework Documentation*. Fonte: https://citrusframework.org/docs/home/

Delaney, K. (2000). *Inside Microsoft SQL Server 2000.*

DeMillo, R. A., Lipton, R. J., & Sayward, F. G. (1978). *Hints on test data selection: Help for the practicing programmer.* IEEE Computer.

Dooley, J. (2011). *Software Development and Professional Practice.*

Droettboom, M. (2015). *Understanding JSON Schema*. Fonte: https://json-schema.org/understanding-json-schema/

ECMA-262. (2018). *ECMAScript® 2018*. Fonte: https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

ECMA-334. (2017). *C# Language.*

ECMA-404. (10 de 07 de 2017). *The JSON Data Interchange Syntax*. Fonte: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

Fielding, R. (1999). *Hypertext Transfer Protocol.*

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Fonte: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* CALIFORNIA. Fonte: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

FOWLER, M. (2010). *Richardson Maturity Model: steps toward the glory of REST.*

G. Booch, J. R. (2005). *UML, Guia do Usuário.* Campus. Fonte: G. Booch, J. Rumbaugh, I. Jacobson.

Hirsch, F., Kemp, J., & Ilkka, J. (2007). *Mobile Web Services: Architecture and Implementation.* John Wiley & Sons.

Howden, W. E. (1987). *Software Engineering and Technology: Functional Program Testing and Analysis.* New York: McGrall-Hill Book Co.

HTTP Status Codes. (23 de 07 de 2017). Fonte: https://httpstatuses.com/

Jakl, M. (2005). *REST Representational State Transfer.* Vienna.

Koskela, L. (2007). *Test Driven: TDD and Acceptance TDD for Java Developers.*

Liberty, J., & MacDonald, B. (2010). *C# Language Fundamentals.*

Microsoft. (2014). *Model-View-Controller*. Fonte: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649643(v=pandp.10)

Morisson, G. R. (2010). *Composição de serviços Web semânticos.* JUIZ DE FORA.

Myers, G. J. (1979). *The Art of Software Testing.* New York: Wiley.

Myers, G. J. (2004). *The Art of Software Testing.* John Wiley & Sons, Inc. .

Paw. (10 de 07 de 2018). *Paw Documentation*. Fonte: https://paw.cloud/docs/

Postman Learning Center. (10 de 07 de 2018). *Postman Docs*. Fonte: https://learning.getpostman.com/docs

Pressman, R. S. (1997). *Software Engineering – A Practitioner's Approach.* McGraw-Hill.

PRESSMAN, R. S. (2005). *Software Engineering: a Practitioner's Approach.* McGraw-Hill.

Rapps, S., & Weyuker, E. J. (1985). *Selecting software test data using data flow information.*

Rauschmayer, D. A. (2014). *Speaking JavaScript: An In-Depth Guide for Programmers*. Fonte: http://speakingjs.com/es5/index.html

Reenskaug, T., & Coplien, J. (2009). *The DCI Architecture: A New Vision of Object-Oriented Programming.*

RestSharp. (10 de 07 de 2018). *RestSharp Wiki*. Fonte: https://github.com/restsharp/RestSharp/wiki

RoboHydra. (10 de 07 de 2018). *RoboHydra Documentation*. Fonte: http://robohydra.org/docs/0.6.8/

Rodrigues, L. C. (Julho de 2009). *Arquitetura REST.* Juiz de Fora.

Saiedian, H., & Janzen, D. (2005). *Test-driven development: Concepts, taxonomy, and future direction. Computer.*

Sampaio, L. F., & Silva, K. A. (2014). *Desenvolvimento de um Web Service baseado em REST.*

Silberschatz, A., Sundarshan, S., & Kort, H. (2016). *Sistema de banco de dados.*

Silva, J., Rocha, C., & Gonçalves, A. (2013). *Uma arquitetura de serviços web.*

Smartbear. (10 de 07 de 2018). *Get Started with Functional API Testing*. Fonte: https://www.soapui.org/docs/functional-testing/getting-started.html

Sommerville, I. (2011). *Software Engineering.*

Souza, S. R. (1996). *Avaliação do custo e eficácia do critério análise de mutantes na atividade de teste de programas.* São Carlos: ICMC-USP.

W3C. (2000). *Simple Object Access Protocol (SOAP) 1.1.* Fonte: https://www.researchgate.net/profile/Satish_Thatte/publication/239553871_Simple_object_access_protocol_SOAP_11/links/54489e4e0cf2f14fb8142a59/Simple-object-access-protocol-SOAP-11.pdf

W3C. (2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Fonte: https://www.w3.org/TR/xml/

WSA. (11 de Fevereiro de 2004). *Web Services Architecture*. Fonte: https://www.w3.org/TR/ws-arch/

ANEXO A – ARTIGO

Desenvolvimento de ferramenta para o teste de requisições REST

Filipe Linemburger, Thiago Mohr da Silveira
Universidade Federal de Santa Catarina (UFSC)
Departamento de Informática e Estatística
Campus Universitário - Florianópolis - Brazil

{filipelinemburger, thiagomohrs}@gmail.com

Abstract. *Quality in systems development is a huge challenge because of the high complexity of current systems, involving human, technical, business and political issues. In the common development scenario manual tests are often done to verify that everything is working according to the specification, and it is normal to find defects. These manual tests are fast and essential, but the execution and repetition of an extensive set of manual tests is a very burdensome and tiresome task. This work of course completion had the objective of solving the problem of repetitiveness in the large number of test cases that need to be executed in applications that use the REST architectural style.*

Resumo. *A qualidade no desenvolvimento dos sistemas é um enorme desafio mediante a alta complexidade dos atuais sistemas desenvolvidos, envolvendo questões humanas, técnicas, de negócio e políticas. No cenário comum de desenvolvimento muitas vezes são feitos testes manuais para verificar se tudo está funcionando conforme a especificação, sendo normal o encontro de defeitos. Esses testes manuais* são *rápidos e essenciais, mas a execução e repetição de um extenso conjunto de testes manuais é uma tarefa muito onerosa e cansativa. Este trabalho de conclusão de curso teve como objetivo resolver o problema da repetitividade na grande quantidade de casos de testes que necessitam serem executados em aplicações que utilizam o estilo arquitetural REST*

1. INTRODUÇÃO

Com a revolução digital, que foi concebida pela disseminação de máquinas (computadores) digitais e do armazenamento de informações de modo digital, deu-se início o período histórico conhecido como Era da Informação.

Nos últimos anos com a crescente utilização de serviços web, foram criados dois tipos de serviços web, que podem ser baseados na arquitetura SOAP (Simple Object Access Protocol) ou em arquitetura REST (Representational State Transfer).

O SOAP é um protocolo baseado em XML que permite a troca de mensagens estruturadas com outras aplicações, em geral utiliza meios genéricos para a realização dessa troca.

O REST é uma arquitetura mais flexível que pode ser incorporada a projetos de aplicações distribuídas, quando utilizado para a implementação de serviços web recebe o nome de RESTFUL e utiliza de diversas linguagens de padronização e encapsulamento de dados, cujo transporte é feito por intermédio do protocolo HTTP, tais como JSON, XML e TEXT (Silva, Rocha, & Gonçalves, 2013), assim uma aplicação pode invocar outra para efetuar tarefas simples ou complexas mesmo que as duas aplicações estejam em diferentes sistemas e escritas em linguagens diferentes. Com o surgimento dessa nova arquitetura de micro serviços, sente-se necessário realizar testes de requisições em serviços REST (Sampaio & Silva, 2014).

Com a adoção de serviços REST alinhados à estratégia de serviços Web, que cada vez mais se tornam amplamente difundidos pela comunidade de desenvolvedores, percebemos nos resultados do desenvolvimento a geração de diversos artefatos independentes. Em função de diversas APIs geradas constantemente, cada vez mais se faz necessário a realização de testes destas APIs para a detecção da ocorrência de erros no processo de construção do programa.

Dito isto, este trabalho apresenta uma proposta de modelo para uma ferramenta web para ajudar na realização de testes em outras ferramentas que utilizam requisição REST. Assim podem ser identificados problemas, que comumente passam despercebidas pelos indivíduos envolvidos no desenvolvimento.

2. OBJETIVOS

Este trabalho tem como objetivo projetar um software (projeto de banco de dados, projeto de casos de uso, projeção total da arquitetura de software, desenvolvimento e execução da aplicação WEB), desenvolvido em linguagem de programação C-Sharp (C#) que testa a qualidade dos dados obtidos a partir de disparos de requisições REST. A ideia de construir tal projeto é motivada pela necessidade de haver um software com funções objetivas e práticas para realização de testes em Interface de Programação de Aplicativos (API). De maneira geral, também objetiva-se aprofundar conhecimentos sobre arquiteturas REST e propor uma mais simplificada, objetiva e prática para realização de testes em APIs.

3. MOTIVAÇÃO

O desenvolvimento de software está cada vez mais utilizando web services para a comunicação entre serviços, plataformas e softwares. Visto isso, é notável a necessidade de ferramentas que façam testes de web services. Após uma pesquisa inicial, não foram encontradas ferramentas que resolvam esses problemas de forma simples, gratuita e com interface gráfica amigável. No cenário atual existem ferramentas extremamente complexas e que necessitam de conhecimento em desenvolvimento de softwares para criação de testes de web services. Além disso existem também ferramentas com nichos

específicos, como frameworks implementados para uso em linguagens especificas, ou então frameworks que fazem testes de uma determinada linguagem.

Este cenário motivou a proposta deste trabalho de conclusão de curso, a criação de uma ferramenta de fácil usabilidade, gratuita, que esteja disponível na internet, sem a necessidade de efetuar instalação em qualquer tipo de dispositivos, e que efetue testes em um grande número de ferramentas, independentemente da linguagem em que foi desenvolvida.

4. PROPOSTA

A proposta consiste em desenvolver uma ferramenta que tem por objetivo auxiliar na validação dos dados obtidos em uma requisição para um web service REST. Sua principal funcionalidade é o cadastro e execução dos testes.

O projeto foi desenvolvido utilizando a linguagem C#. Entende-se que esta linguagem atende completamente os requisitos, por ser uma das mais usadas no mundo para a programação web. De fácil integração com os webservices e atendendo a necessidade por uma manutenibilidade mais acessível.

O banco de dados utilizado foi o SQLserver. Esse banco de dados foi escolhido porque apresenta boa performance e pode ser utilizado gratuitamente, preenchendo os requisitos necessários para o projeto.

Os requisitos funcionais são as funções fornecidas pelo sistema. Descrevem a reação do sistema dadas determinadas entradas específicas e também como o sistema deverá se comportar sob as mais diversas situações. Em alguns casos, os requisitos funcionais podem descrever o que o sistema não deve fazer.

Abaixo estão listados os requisitos funcionais identificados para o funcionamento adequado desta ferramenta:

6. Cadastro de usuário. Informações referentes ao usuário do sistema, com os atributos nome, e-mail, curso, login e senha.
7. Autenticação. O sistema deve ter como tela inicial espaço para o usuário inserir login e senha. Também deve estar presente a opção para cadastro de novo usuário.
8. Cadastro de teste. Principal funcionalidade do sistema. Conterá informações para execução dos testes e a validação dos dados obtidos.
9. Lista de testes. O sistema deve exibir uma lista com os testes já cadastrados, permitindo sua edição e exclusão.
10. Verificação de resultados. O sistema deve exibir as informações referentes às avaliações obtidas a partir da execução de um teste.

Os requisitos não-funcionais são restrições que agem sobre as funcionalidades providas pelo sistema. O sistema, por se tratar de um sistema web deverá rodar nos navegadores elencados neste requisito. O comportamento deve ser o mesmo, no que se refere às funcionalidades. Segue lista dos requisitos não funcionais deste sistema:

2. Compatibilidade. O sistema deve ser compatível com os navegadores Chrome, Firefox, Edge e Safari.

O diagrama de casos de uso documenta o que o novo sistema faz do ponto de vista dos usuários descrevendo as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema (G. Booch, 2005). Apenas foram descritas as principais funcionalidades do sistema. Abaixo podemos compreender melhor como são os casos de uso:

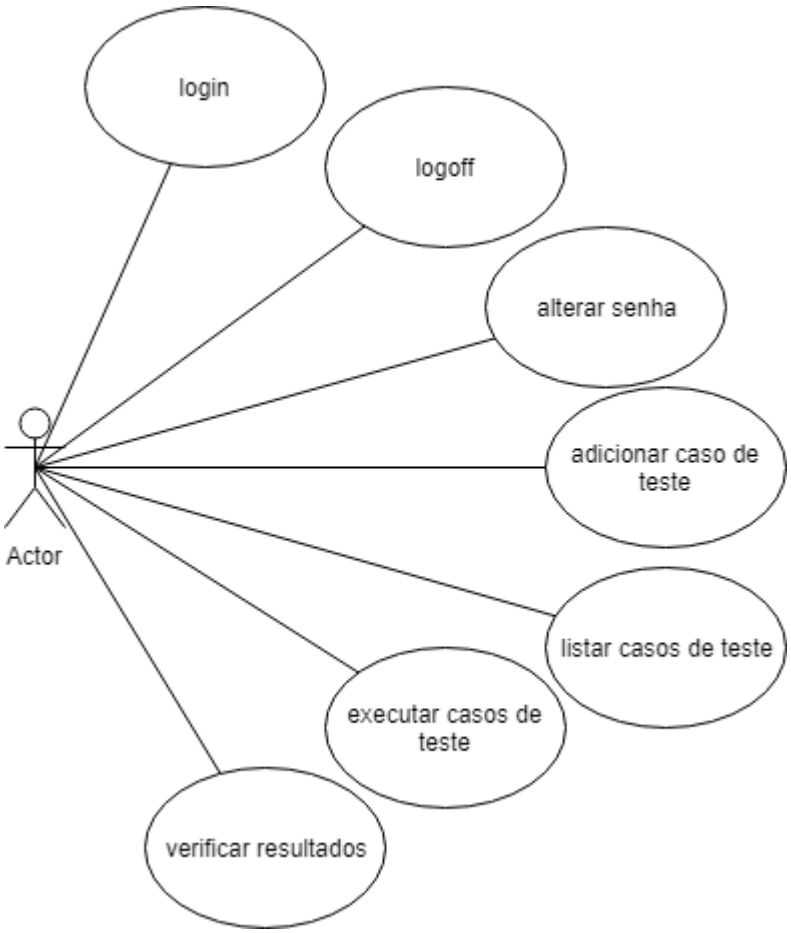


Figura 18 - Diagrama de casos de uso

Foi desenvolvido um protótipo de interface de usuário de baixa fidelidade onde é possível de forma mais rápida e econômica definir e experimentar um projeto, onde deverão estar dispostas na interface as informações da aplicação. Apenas durante a interação real do usuário com o sistema que os detalhes realmente são percebidos (PRESSMAN, 2005).

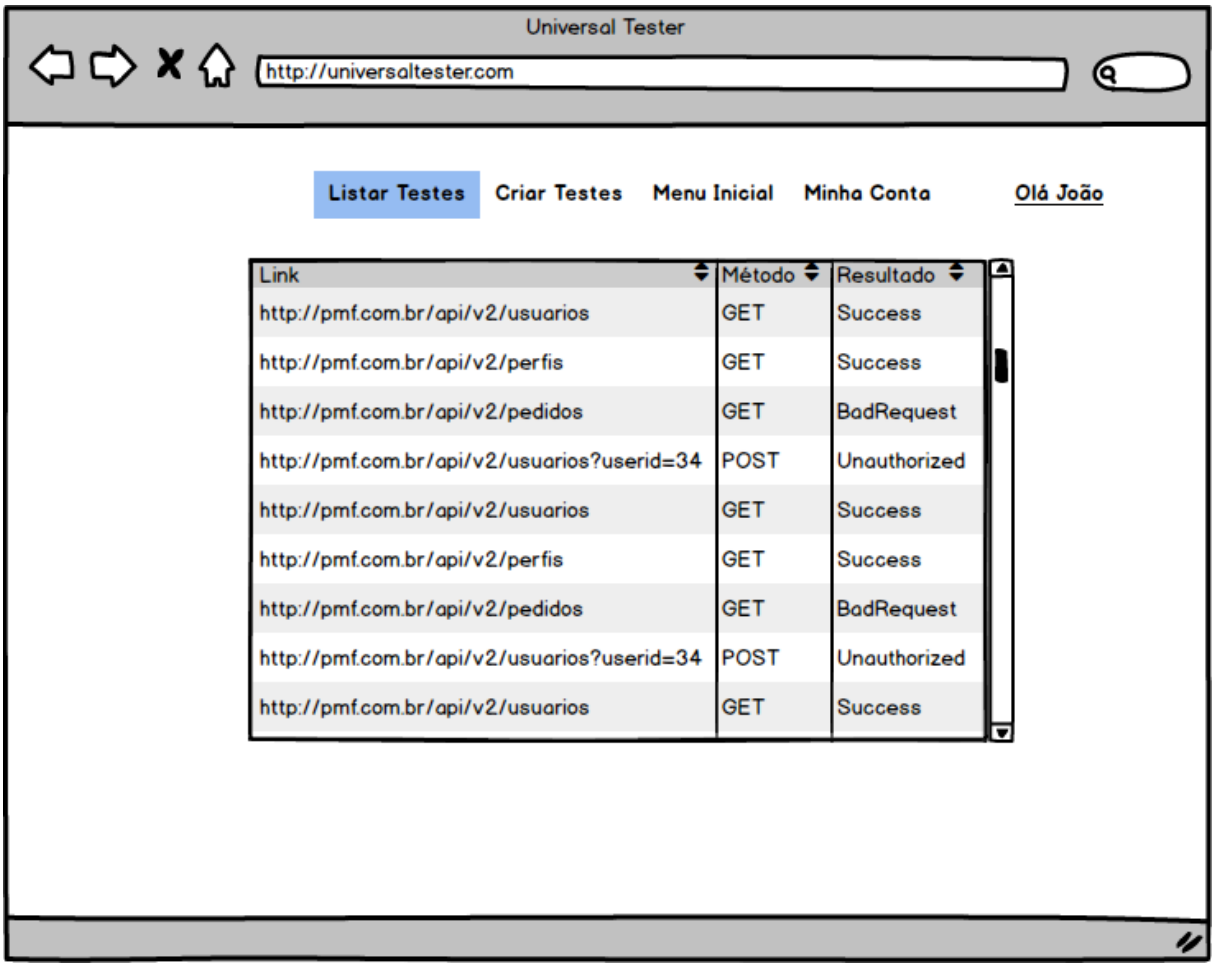


Figura 19 - Protótipo de baixa fidelidade da interface de usuário

Em função da ferramenta desenvolvida armazenar dados de usuário e dos testes de requisições cadastradas, houve a necessidade de utilizar um banco de dados. A modelagem da estrutura deste banco de dados está no diagrama abaixo.

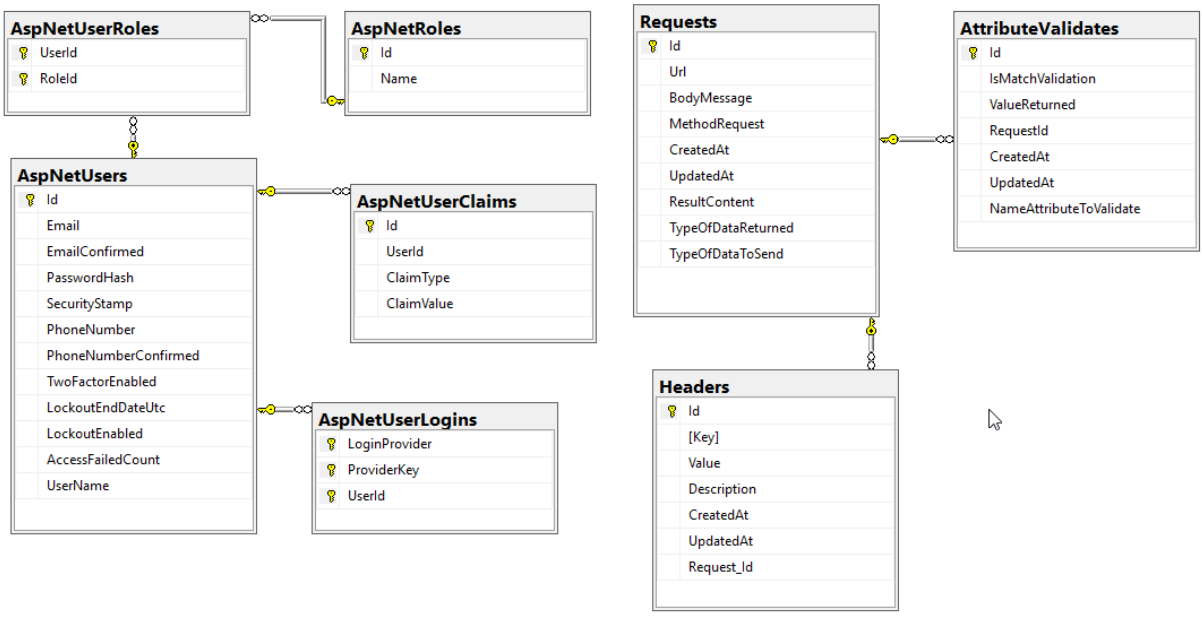


Figura 20 - Diagrama do banco de dados

Na tela de cadastro de novo teste de requisição é necessário inserir a URL da requisição, o método HTTP utilizado no teste, o nome do atributo a ser validado dentro da resposta em formato JSON, o tipo e valor do atributo a ser validado, para assim poder ser validado e assumir que o teste criado está correto.

Após o desenvolvimento da ferramenta, a aparência final da interface a ser utilizada pelo usuário para cadastrar e gerenciar os testes a serem feitos em web services ficou conforme a imagem abaixo.

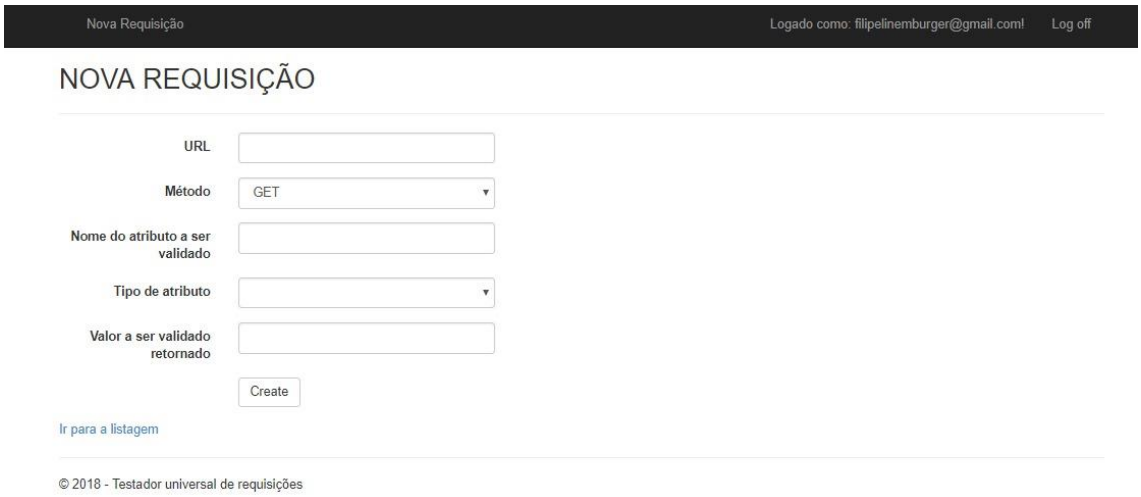


Figura 21 - Tela de cadastro de novo teste de requisição

5. CONCLUSÃO

Este trabalho resultou na criação de um sistema web eficiente para realização de testes para requisições REST, proporcionando uma maior rapidez no teste destas requisições e possibilitando a avaliação dos mesmos.

Como base de comparação, o trabalho desenvolvido neste TCC tem como grande destaque o mapeamento de campos através da linguagem especificada de um determinado campo dentro de uma estrutura JSON Object

Com este trabalho foi desenvolvido uma ferramenta eficiente para realização de testes para requisições REST, proporcionando uma maior rapidez no teste destas requisições, uma vez que é possível cadastrar diversos testes e executá-los quantas vezes quiser apenas com um clique.

A ferramenta desenvolvida implica que exista uma definição do teste, que deve ser cadastrado na ferramenta, explicitando o tipo da resposta e os valores que devem ser retornados na resposta da requisição. Durante a execução dos testes são validados o tipo da resposta da requisição e o valor específico do atributo da resposta.

Com a finalização do desenvolvimento foi averiguado a melhoria no tempo de execução de testes dado que agora todos os testes são cadastrados apenas uma vez e executados diversas vezes.

REFERENCIAS

G. Booch, J. R. (2005). *UML, Guia do Usuário.* Campus. Fonte: G. Booch, J. Rumbaugh, I. Jacobson.

Pressman, R. S. (1997). *Software Engineering – A Practitioner's Approach.* McGraw-Hill.

Sampaio, L. F., & Silva, K. A. (2014). *Desenvolvimento de um Web Service baseado em REST.*

Silva, J., Rocha, C., & Gonçalves, A. (2013). *Uma arquitetura de serviços web.*

ANEXO B – CÓDIGO FONTE DA FERRAMENTA

AccountController.cs

```
using System.Linq;
using System.Threading.Tasks;
using System.Web;
using System.Web.Mvc;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin.Security;
using Universal_Tester.Models;

namespace Universal_Tester.Controllers
{
    [Authorize]
    public class AccountController : Controller
    {
        private ApplicationSignInManager _signInManager;
        private ApplicationUserManager _userManager;

        public AccountController()
        {
        }

        public AccountController(ApplicationUserManager userManager,
ApplicationSignInManager signInManager )
        {
            UserManager = userManager;
            SignInManager = signInManager;
        }

        public ApplicationSignInManager SignInManager
        {
            get
            {
                return _signInManager ??
HttpContext.GetOwinContext().Get<ApplicationSignInManager>();
            }
            private set
            {
                _signInManager = value;
            }
        }

        public ApplicationUserManager UserManager
        {
            get
            {
                return _userManager ??
HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();
            }
            private set
            {
                _userManager = value;
            }
        }
```

```
        //
        // GET: /Account/Login
        [AllowAnonymous]
        public ActionResult Login(string returnUrl)
        {
            ViewBag.ReturnUrl = returnUrl;
            return View();
        }

        //
        // POST: /Account/Login
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)
        {
            if (!ModelState.IsValid)
            {
                return View(model);
            }

            // This doesn't count login failures towards account lockout
            // To enable password failures to trigger account lockout, change to
shouldLockout: true
            var result = await SignInManager.PasswordSignInAsync(model.Email,
model.Password, model.RememberMe, shouldLockout: false);
            switch (result)
            {
                case SignInStatus.Success:
                    return RedirectToLocal(returnUrl);
                case SignInStatus.LockedOut:
                    return View("Lockout");
                case SignInStatus.RequiresVerification:
                    return RedirectToAction("SendCode", new { ReturnUrl = returnUrl,
RememberMe = model.RememberMe });
                case SignInStatus.Failure:
                default:
                    ModelState.AddModelError("", "Invalid login attempt.");
                    return View(model);
            }
        }

        //
        // GET: /Account/VerifyCode
        [AllowAnonymous]
        public async Task<ActionResult> VerifyCode(string provider, string returnUrl, bool
rememberMe)
        {
            // Require that the user has already logged in via username/password or
external login
            if (!await SignInManager.HasBeenVerifiedAsync())
            {
                return View("Error");
```

```
            }
            return View(new VerifyCodeViewModel { Provider = provider, ReturnUrl =
returnUrl, RememberMe = rememberMe });
        }

        //
        // POST: /Account/VerifyCode
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> VerifyCode(VerifyCodeViewModel model)
        {
            if (!ModelState.IsValid)
            {
                return View(model);
            }

            // The following code protects for brute force attacks against the two factor
codes.
            // If a user enters incorrect codes for a specified amount of time then the user
account
            // will be locked out for a specified amount of time.
            // You can configure the account lockout settings in IdentityConfig
            var result = await SignInManager.TwoFactorSignInAsync(model.Provider,
model.Code, isPersistent:  model.RememberMe, rememberBrowser:
model.RememberBrowser);
            switch (result)
            {
                case SignInStatus.Success:
                    return RedirectToLocal(model.ReturnUrl);
                case SignInStatus.LockedOut:
                    return View("Lockout");
                case SignInStatus.Failure:
                default:
                    ModelState.AddModelError("", "Invalid code.");
                    return View(model);
            }
        }

        //
        // GET: /Account/Register
        [AllowAnonymous]
        public ActionResult Register()
        {
            return View();
        }

        //
        // POST: /Account/Register
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Register(RegisterViewModel model)
        {
```

```
            if (ModelState.IsValid)
            {
                var user = new ApplicationUser { UserName = model.Email, Email =
model.Email };
                var result = await UserManager.CreateAsync(user, model.Password);
                if (result.Succeeded)
                {
                    await SignInManager.SignInAsync(user, isPersistent:false,
rememberBrowser:false);

                    // For more information on how to enable account confirmation and
password reset please visit https://go.microsoft.com/fwlink/?LinkID=320771
                    // Send an email with this link
                    // string code = await
UserManager.GenerateEmailConfirmationTokenAsync(user.Id);
                    // var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId =
user.Id, code = code }, protocol: Request.Url.Scheme);
                    // await UserManager.SendEmailAsync(user.Id, "Confirm your account",
"Please confirm your account by clicking <a href=\"" + callbackUrl + "\">here</a>");

                    return RedirectToAction("Index", "Home");
                }
                AddErrors(result);
            }

            // If we got this far, something failed, redisplay form
            return View(model);
        }

        //
        // GET: /Account/ConfirmEmail
        [AllowAnonymous]
        public async Task<ActionResult> ConfirmEmail(string userId, string code)
        {
            if (userId == null || code == null)
            {
                return View("Error");
            }
            var result = await UserManager.ConfirmEmailAsync(userId, code);
            return View(result.Succeeded ? "ConfirmEmail" : "Error");
        }

        //
        // GET: /Account/ForgotPassword
        [AllowAnonymous]
        public ActionResult ForgotPassword()
        {
            return View();
        }

        //
        // POST: /Account/ForgotPassword
        [HttpPost]
        [AllowAnonymous]
```

```
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> ForgotPassword(ForgotPasswordViewModel
model)
        {
            if (ModelState.IsValid)
            {
                var user = await UserManager.FindByNameAsync(model.Email);
                if (user == null || !(await UserManager.IsEmailConfirmedAsync(user.Id)))
                {
                    // Don't reveal that the user does not exist or is not confirmed
                    return View("ForgotPasswordConfirmation");
                }

                // For more information on how to enable account confirmation and password
reset please visit https://go.microsoft.com/fwlink/?LinkID=320771
                // Send an email with this link
                // string code = await
UserManager.GeneratePasswordResetTokenAsync(user.Id);
                // var callbackUrl = Url.Action("ResetPassword", "Account", new { userId =
user.Id, code = code }, protocol: Request.Url.Scheme);
                // await UserManager.SendEmailAsync(user.Id, "Reset Password", "Please
reset your password by clicking <a href=\"" + callbackUrl + "\">here</a>");
                // return RedirectToAction("ForgotPasswordConfirmation", "Account");
            }

            // If we got this far, something failed, redisplay form
            return View(model);
        }

        //
        // GET: /Account/ForgotPasswordConfirmation
        [AllowAnonymous]
        public ActionResult ForgotPasswordConfirmation()
        {
            return View();
        }

        //
        // GET: /Account/ResetPassword
        [AllowAnonymous]
        public ActionResult ResetPassword(string code)
        {
            return code == null ? View("Error") : View();
        }

        //
        // POST: /Account/ResetPassword
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> ResetPassword(ResetPasswordViewModel
model)
        {
            if (!ModelState.IsValid)
```

```
            {
                return View(model);
            }
            var user = await UserManager.FindByNameAsync(model.Email);
            if (user == null)
            {
                // Don't reveal that the user does not exist
                return RedirectToAction("ResetPasswordConfirmation", "Account");
            }
            var result = await UserManager.ResetPasswordAsync(user.Id, model.Code,
model.Password);
            if (result.Succeeded)
            {
                return RedirectToAction("ResetPasswordConfirmation", "Account");
            }
            AddErrors(result);
            return View();
        }

        //
        // GET: /Account/ResetPasswordConfirmation
        [AllowAnonymous]
        public ActionResult ResetPasswordConfirmation()
        {
            return View();
        }

        //
        // POST: /Account/ExternalLogin
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public ActionResult ExternalLogin(string provider, string returnUrl)
        {
            // Request a redirect to the external login provider
            return new ChallengeResult(provider, Url.Action("ExternalLoginCallback",
"Account", new { ReturnUrl = returnUrl }));
        }

        //
        // GET: /Account/SendCode
        [AllowAnonymous]
        public async Task<ActionResult> SendCode(string returnUrl, bool rememberMe)
        {
            var userId = await SignInManager.GetVerifiedUserIdAsync();
            if (userId == null)
            {
                return View("Error");
            }
            var userFactors = await
UserManager.GetValidTwoFactorProvidersAsync(userId);
            var factorOptions = userFactors.Select(purpose => new SelectListItem { Text =
purpose, Value = purpose }).ToList();
            return View(new SendCodeViewModel { Providers = factorOptions, ReturnUrl =
returnUrl, RememberMe = rememberMe });
```

```
        }

        //
        // POST: /Account/SendCode
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> SendCode(SendCodeViewModel model)
        {
            if (!ModelState.IsValid)
            {
                return View();
            }

            // Generate the token and send it
            if (!await SignInManager.SendTwoFactorCodeAsync(model.SelectedProvider))
            {
                return View("Error");
            }
            return RedirectToAction("VerifyCode", new { Provider =
model.SelectedProvider, ReturnUrl = model.ReturnUrl, RememberMe =
model.RememberMe });
        }

        //
        // GET: /Account/ExternalLoginCallback
        [AllowAnonymous]
        public async Task<ActionResult> ExternalLoginCallback(string returnUrl)
        {
            var loginInfo = await AuthenticationManager.GetExternalLoginInfoAsync();
            if (loginInfo == null)
            {
                return RedirectToAction("Login");
            }

            // Sign in the user with this external login provider if the user already has a login
            var result = await SignInManager.ExternalSignInAsync(loginInfo, isPersistent:
false);
            switch (result)
            {
                case SignInStatus.Success:
                    return RedirectToLocal(returnUrl);
                case SignInStatus.LockedOut:
                    return View("Lockout");
                case SignInStatus.RequiresVerification:
                    return RedirectToAction("SendCode", new { ReturnUrl = returnUrl,
RememberMe = false });
                case SignInStatus.Failure:
                default:
                    // If the user does not have an account, then prompt the user to create an
account
                    ViewBag.ReturnUrl = returnUrl;
                    ViewBag.LoginProvider = loginInfo.Login.LoginProvider;
```

```
                return View("ExternalLoginConfirmation", new
ExternalLoginConfirmationViewModel { Email = loginInfo.Email });
            }
        }

        //
        // POST: /Account/ExternalLoginConfirmation
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
ExternalLoginConfirmation(ExternalLoginConfirmationViewModel model, string
returnUrl)
        {
            if (User.Identity.IsAuthenticated)
            {
                return RedirectToAction("Index", "Manage");
            }

            if (ModelState.IsValid)
            {
                // Get the information about the user from the external login provider
                var info = await AuthenticationManager.GetExternalLoginInfoAsync();
                if (info == null)
                {
                    return View("ExternalLoginFailure");
                }
                var user = new ApplicationUser { UserName = model.Email, Email =
model.Email };
                var result = await UserManager.CreateAsync(user);
                if (result.Succeeded)
                {
                    result = await UserManager.AddLoginAsync(user.Id, info.Login);
                    if (result.Succeeded)
                    {
                        await SignInManager.SignInAsync(user, isPersistent: false,
rememberBrowser: false);
                        return RedirectToLocal(returnUrl);
                    }
                }
                AddErrors(result);
            }

            ViewBag.ReturnUrl = returnUrl;
            return View(model);
        }

        //
        // POST: /Account/LogOff
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult LogOff()
        {
```

```
            AuthenticationManager.SignOut(DefaultAuthenticationTypes.ApplicationCookie
);
            return RedirectToAction("Index", "Home");
        }

        //
        // GET: /Account/ExternalLoginFailure
        [AllowAnonymous]
        public ActionResult ExternalLoginFailure()
        {
            return View();
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing)
            {
                if (_userManager != null)
                {
                    _userManager.Dispose();
                    _userManager = null;
                }

                if (_signInManager != null)
                {
                    _signInManager.Dispose();
                    _signInManager = null;
                }
            }

            base.Dispose(disposing);
        }

        #region Helpers
        // Used for XSRF protection when adding external logins
        private const string XsrfKey = "XsrfId";

        private IAuthenticationManager AuthenticationManager
        {
            get
            {
                return HttpContext.GetOwinContext().Authentication;
            }
        }

        private void AddErrors(IdentityResult result)
        {
            foreach (var error in result.Errors)
            {
                ModelState.AddModelError("", error);
            }
        }

        private ActionResult RedirectToLocal(string returnUrl)
```

```
        {
            if (Url.IsLocalUrl(returnUrl))
            {
                return Redirect(returnUrl);
            }
            return RedirectToAction("Index", "Home");
        }

        internal class ChallengeResult : HttpUnauthorizedResult
        {
            public ChallengeResult(string provider, string redirectUri)
                : this(provider, redirectUri, null)
            {
            }

            public ChallengeResult(string provider, string redirectUri, string userId)
            {
                LoginProvider = provider;
                RedirectUri = redirectUri;
                UserId = userId;
            }

            public string LoginProvider { get; set; }
            public string RedirectUri { get; set; }
            public string UserId { get; set; }

            public override void ExecuteResult(ControllerContext context)
            {
                var properties = new AuthenticationProperties { RedirectUri = RedirectUri };
                if (UserId != null)
                {
                    properties.Dictionary[XsrfKey] = UserId;
                }
                context.HttpContext.GetOwinContext().Authentication.Challenge(properties,
LoginProvider);
            }
        }
        #endregion
    }
}
```

HomeController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Universal_Tester.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
```

```
        {
            if (!User.Identity.IsAuthenticated)
                return RedirectToAction("Login", "Account");
            return View();
        }

        public ActionResult About()
        {
            ViewBag.Message = "Your application description page.";

            return View();
        }

        public ActionResult Contact()
        {
            ViewBag.Message = "Your contact page.";

            return View();
        }
        public ActionResult Login()
        {
            return View();
        }
    }
}
```

ManageController.cs

```
using System;
using System.Linq;
using System.Threading.Tasks;
using System.Web;
using System.Web.Mvc;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin.Security;
using Universal_Tester.Models;

namespace Universal_Tester.Controllers
{
    [Authorize]
    public class ManageController : Controller
    {
        private ApplicationSignInManager _signInManager;
        private ApplicationUserManager _userManager;

        public ManageController()
        {
        }

        public ManageController(ApplicationUserManager userManager,
ApplicationSignInManager signInManager)
        {
            UserManager = userManager;
```

```
            SignInManager = signInManager;
        }

        public ApplicationSignInManager SignInManager
        {
            get
            {
                return _signInManager ??
HttpContext.GetOwinContext().Get<ApplicationSignInManager>();
            }
            private set
            {
                _signInManager = value;
            }
        }

        public ApplicationUserManager UserManager
        {
            get
            {
                return _userManager ??
HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();
            }
            private set
            {
                _userManager = value;
            }
        }

        //
        // GET: /Manage/Index
        public async Task<ActionResult> Index(ManageMessageId? message)
        {
            ViewBag.StatusMessage =
                message == ManageMessageId.ChangePasswordSuccess ? "Your
password has been changed."
                : message == ManageMessageId.SetPasswordSuccess ? "Your password
has been set."
                : message == ManageMessageId.SetTwoFactorSuccess ? "Your two-factor
authentication provider has been set."
                : message == ManageMessageId.Error ? "An error has occurred."
                : message == ManageMessageId.AddPhoneSuccess ? "Your phone number
was added."
                : message == ManageMessageId.RemovePhoneSuccess ? "Your phone
number was removed."
                : "";

            var userId = User.Identity.GetUserId();
            var model = new IndexViewModel
            {
                HasPassword = HasPassword(),
                PhoneNumber = await UserManager.GetPhoneNumberAsync(userId),
                TwoFactor = await UserManager.GetTwoFactorEnabledAsync(userId),
                Logins = await UserManager.GetLoginsAsync(userId),
```

```
            BrowserRemembered = await
AuthenticationManager.TwoFactorBrowserRememberedAsync(userId)
        };
        return View(model);
    }

    //
    // POST: /Manage/RemoveLogin
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<ActionResult> RemoveLogin(string loginProvider, string
providerKey)
    {
        ManageMessageId? message;
        var result = await UserManager.RemoveLoginAsync(User.Identity.GetUserId(),
new UserLoginInfo(loginProvider, providerKey));
        if (result.Succeeded)
        {
            var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());
            if (user != null)
            {
                await SignInManager.SignInAsync(user, isPersistent: false,
rememberBrowser: false);
            }
            message = ManageMessageId.RemoveLoginSuccess;
        }
        else
        {
            message = ManageMessageId.Error;
        }
        return RedirectToAction("ManageLogins", new { Message = message });
    }

    //
    // GET: /Manage/AddPhoneNumber
    public ActionResult AddPhoneNumber()
    {
        return View();
    }

    //
    // POST: /Manage/AddPhoneNumber
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<ActionResult>
AddPhoneNumber(AddPhoneNumberViewModel model)
    {
        if (!ModelState.IsValid)
        {
            return View(model);
        }
        // Generate the token and send it
        var code = await
UserManager.GenerateChangePhoneNumberTokenAsync(User.Identity.GetUserId(),
model.Number);
```

```
            if (UserManager.SmsService != null)
            {
                var message = new IdentityMessage
                {
                    Destination = model.Number,
                    Body = "Your security code is: " + code
                };
                await UserManager.SmsService.SendAsync(message);
            }
            return RedirectToAction("VerifyPhoneNumber", new { PhoneNumber =
model.Number });
        }

        //
        // POST: /Manage/EnableTwoFactorAuthentication
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> EnableTwoFactorAuthentication()
        {
            await UserManager.SetTwoFactorEnabledAsync(User.Identity.GetUserId(),
true);
            var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());
            if (user != null)
            {
                await SignInManager.SignInAsync(user, isPersistent: false,
rememberBrowser: false);
            }
            return RedirectToAction("Index", "Manage");
        }

        //
        // POST: /Manage/DisableTwoFactorAuthentication
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> DisableTwoFactorAuthentication()
        {
            await UserManager.SetTwoFactorEnabledAsync(User.Identity.GetUserId(),
false);
            var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());
            if (user != null)
            {
                await SignInManager.SignInAsync(user, isPersistent: false,
rememberBrowser: false);
            }
            return RedirectToAction("Index", "Manage");
        }

        //
        // GET: /Manage/VerifyPhoneNumber
        public async Task<ActionResult> VerifyPhoneNumber(string phoneNumber)
        {
            var code = await
UserManager.GenerateChangePhoneNumberTokenAsync(User.Identity.GetUserId(),
phoneNumber);
            // Send an SMS through the SMS provider to verify the phone number
```

```
            return phoneNumber == null ? View("Error") : View(new
VerifyPhoneNumberViewModel { PhoneNumber = phoneNumber });
        }

        //
        // POST: /Manage/VerifyPhoneNumber
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult>
VerifyPhoneNumber(VerifyPhoneNumberViewModel model)
        {
            if (!ModelState.IsValid)
            {
                return View(model);
            }
            var result = await
UserManager.ChangePhoneNumberAsync(User.Identity.GetUserId(),
model.PhoneNumber, model.Code);
            if (result.Succeeded)
            {
                var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());
                if (user != null)
                {
                    await SignInManager.SignInAsync(user, isPersistent: false,
rememberBrowser: false);
                }
                return RedirectToAction("Index", new { Message =
ManageMessageId.AddPhoneSuccess });
            }
            // If we got this far, something failed, redisplay form
            ModelState.AddModelError("", "Failed to verify phone");
            return View(model);
        }

        //
        // POST: /Manage/RemovePhoneNumber
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> RemovePhoneNumber()
        {
            var result = await
UserManager.SetPhoneNumberAsync(User.Identity.GetUserId(), null);
            if (!result.Succeeded)
            {
                return RedirectToAction("Index", new { Message = ManageMessageId.Error
});
            }
            var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());
            if (user != null)
            {
                await SignInManager.SignInAsync(user, isPersistent: false,
rememberBrowser: false);
            }
            return RedirectToAction("Index", new { Message =
ManageMessageId.RemovePhoneSuccess });
```

```
        }

        //
        // GET: /Manage/ChangePassword
        public ActionResult ChangePassword()
        {
            return View();
        }

        //
        // POST: /Manage/ChangePassword
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> ChangePassword(ChangePasswordViewModel
model)
        {
            if (!ModelState.IsValid)
            {
                return View(model);
            }
            var result = await
UserManager.ChangePasswordAsync(User.Identity.GetUserId(), model.OldPassword,
model.NewPassword);
            if (result.Succeeded)
            {
                var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());
                if (user != null)
                {
                    await SignInManager.SignInAsync(user, isPersistent: false,
rememberBrowser: false);
                }
                return RedirectToAction("Index", new { Message =
ManageMessageId.ChangePasswordSuccess });
            }
            AddErrors(result);
            return View(model);
        }

        //
        // GET: /Manage/SetPassword
        public ActionResult SetPassword()
        {
            return View();
        }

        //
        // POST: /Manage/SetPassword
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> SetPassword(SetPasswordViewModel model)
        {
            if (ModelState.IsValid)
            {
```

```
            var result = await
UserManager.AddPasswordAsync(User.Identity.GetUserId(), model.NewPassword);
            if (result.Succeeded)
            {
                var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());
                if (user != null)
                {
                    await SignInManager.SignInAsync(user, isPersistent: false,
rememberBrowser: false);
                }
                return RedirectToAction("Index", new { Message =
ManageMessageId.SetPasswordSuccess });
            }
            AddErrors(result);
        }

        // If we got this far, something failed, redisplay form
        return View(model);
    }

    //
    // GET: /Manage/ManageLogins
    public async Task<ActionResult> ManageLogins(ManageMessageId? message)
    {
        ViewBag.StatusMessage =
            message == ManageMessageId.RemoveLoginSuccess ? "The external login
was removed."
            : message == ManageMessageId.Error ? "An error has occurred."
            : "";
        var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());
        if (user == null)
        {
            return View("Error");
        }
        var userLogins = await
UserManager.GetLoginsAsync(User.Identity.GetUserId());
        var otherLogins =
AuthenticationManager.GetExternalAuthenticationTypes().Where(auth =>
userLogins.All(ul => auth.AuthenticationType != ul.LoginProvider)).ToList();
        ViewBag.ShowRemoveButton = user.PasswordHash != null || userLogins.Count
> 1;
        return View(new ManageLoginsViewModel
        {
            CurrentLogins = userLogins,
            OtherLogins = otherLogins
        });
    }

    //
    // POST: /Manage/LinkLogin
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult LinkLogin(string provider)
    {
```

```
            // Request a redirect to the external login provider to link a login for the current
user
            return new AccountController.ChallengeResult(provider,
Url.Action("LinkLoginCallback", "Manage"), User.Identity.GetUserId());
        }

        //
        // GET: /Manage/LinkLoginCallback
        public async Task<ActionResult> LinkLoginCallback()
        {
            var loginInfo = await
AuthenticationManager.GetExternalLoginInfoAsync(XsrfKey,
User.Identity.GetUserId());
            if (loginInfo == null)
            {
                return RedirectToAction("ManageLogins", new { Message =
ManageMessageId.Error });
            }
            var result = await UserManager.AddLoginAsync(User.Identity.GetUserId(),
loginInfo.Login);
            return result.Succeeded ? RedirectToAction("ManageLogins") :
RedirectToAction("ManageLogins", new { Message = ManageMessageId.Error });
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing && _userManager != null)
            {
                _userManager.Dispose();
                _userManager = null;
            }

            base.Dispose(disposing);
        }

#region Helpers
        // Used for XSRF protection when adding external logins
        private const string XsrfKey = "XsrfId";

        private IAuthenticationManager AuthenticationManager
        {
            get
            {
                return HttpContext.GetOwinContext().Authentication;
            }
        }

        private void AddErrors(IdentityResult result)
        {
            foreach (var error in result.Errors)
            {
                ModelState.AddModelError("", error);
            }
        }
```

```
        private bool HasPassword()
        {
            var user = UserManager.FindById(User.Identity.GetUserId());
            if (user != null)
            {
                return user.PasswordHash != null;
            }
            return false;
        }

        private bool HasPhoneNumber()
        {
            var user = UserManager.FindById(User.Identity.GetUserId());
            if (user != null)
            {
                return user.PhoneNumber != null;
            }
            return false;
        }

        public enum ManageMessageId
        {
            AddPhoneSuccess,
            ChangePasswordSuccess,
            SetTwoFactorSuccess,
            SetPasswordSuccess,
            RemoveLoginSuccess,
            RemovePhoneSuccess,
            Error
        }

#endregion
    }
}
```

RequestController.cs

```
using System;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Net;
using System.Web.Mvc;
using Universal_Tester.Context;
using Universal_Tester.Entities;
using Universal_Tester.Models;
using Universal_Tester.Services;

namespace Universal_Tester.Controllers
{
    public class RequestController : Controller
    {
        private readonly UniversalTesterContext db = new UniversalTesterContext();
```

```
        private readonly RequestsService requestService = new RequestsService();
        // GET: Request
        public ActionResult Index()
        {
            var request = requestService.GetRequestsByUserEmail(User.Identity.Name);
            var retorno = request.Select(x => new RequestViewModel()
            {
                id = x.Id,
                IsMatchValidation = x.AtributteValidate.IsMatchValidation,
                Url = x.Url,
                ResultContent = x.ResultContent,
                ValueReturned = x.AtributteValidate.ValueReturned,
                MethodRequest = x.MethodRequest,
                NameAttributeToValidate = x.AtributteValidate.NameAttributeToValidate,
                IsMatchTypeOfAttribute = x.AtributteValidate.IsMatchTypeOfAttribute,
                TypeOfAttribute = x.AtributteValidate.TypeOfAttribute,
                StatusCode = x.StatusCode.GetHashCode(),
                TimeToExecute = x.TimeToExecute
            });
            return View(retorno.ToList());
        }

        public ActionResult ExecuteAll()
        {
            var requestsList =
requestService.GetRequestsByUserEmail(User.Identity.Name);
            requestService.ExecuteAll(requestsList);
            return RedirectToAction("Index");
        }

        // GET: Request/Details/5
        public ActionResult Details(int? id)
        {
            if (id == null)
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            Request request = requestService.FindRequest(id.Value);
            if (request == null)
                return HttpNotFound();
            return View(request);
        }

        // GET: Request/Create
        public ActionResult Create()
        {
            return View(new RequestViewModel());
        }

        // POST: Request/Create
        // To protect from overposting attacks, please enable the specific properties you
want to bind to, for
        // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Create(RequestViewModel requestViewModel)
```

```
        {
            if (ModelState.IsValid)
            {
                requestService.CreateRequest(requestViewModel,
User.Identity.Name);
                return RedirectToAction("");
            }
            ViewBag.AtributteValidateId = new SelectList(db.AttributeValidate, "Id",
"NameAttributeToValidate", requestViewModel.AtributteValidateId);
            return View(requestViewModel);
        }

        // GET: Request/Edit/5
        [HttpGet]
        public ActionResult Edit(int id)
        {
            var requestViewModel = requestService.FindRequestToEditions(id);
            ViewBag.AtributteValidateId = new SelectList(db.AttributeValidate, "Id",
"NameAttributeToValidate", requestViewModel.AtributteValidateId);
            return View(requestViewModel);
        }

        // POST: Request/Edit/5
        // To protect from overposting attacks, please enable the specific properties you
want to bind to, for
        // more details see https://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult Edit(RequestViewModel requestViewModel)
        {
            if (ModelState.IsValid)
            {
                requestService.SaveEditionsRequest(requestViewModel);
                return RedirectToAction("Index");
            }
            ViewBag.AtributteValidateId = new SelectList(db.AttributeValidate, "Id",
"NameAttributeToValidate", requestViewModel.AtributteValidateId);
            return View(requestViewModel.id);
        }

        // GET: Request/Delete/5
        public ActionResult Delete(int? id)
        {
            if (id == null)
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            Request request = requestService.FindRequest(id.Value);
            if (request == null)
                return HttpNotFound();
            return View(request);
        }

        // POST: Request/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
```

```
        public ActionResult DeleteConfirmed(int id)
        {
            requestService.RemoveRequest(id);
            return RedirectToAction("Index");
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing)
                db.Dispose();
            base.Dispose(disposing);
        }
    }
}
```

AttributeValidate.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using static Universal_Tester.Utils.Enums;

namespace Universal_Tester.Entities
{
    public class AttributeValidate : EntityBase
    {
        public string NameAttributeToValidate { get; set; }
        public bool IsMatchValidation { get; set; }
        public string ValueReturned { get; set; }
        public TypeOfAttribute TypeOfAttribute { get; set; }
        public bool IsMatchTypeOfAttribute { get; set; }

    }
}
```

EntityBase.cs

```
using System;
using System.ComponentModel.DataAnnotations;

namespace Universal_Tester.Entities
{
    public class EntityBase
    {
        public int Id { get; set; }

        [DisplayFormat(DataFormatString = "{0:dd/MM/yyyy hh:mm}")]
        [Display (Name = "Data de criação")]
        public DateTime CreatedAt { get; set; }

        [DisplayFormat(DataFormatString = "{0:dd/MM/yyyy hh:mm}")]
        [Display(Name = "Data de modificação")]
        public DateTime UpdatedAt { get; set; }
    }
```

```
}
```

Header.cs

```
using System.ComponentModel.DataAnnotations;

namespace Universal_Tester.Entities
{
    public class Header : EntityBase
    {
        public Header()
        {

        }
        public Header(string key, string value)
        {
            Key = key;
            Value = value;
        }
        [Display(Name = "Chave")]
        public string Key { get; set; }
        [Display(Name = "Valor")]
        public string Value { get; set; }
        public int RequestId { get; set; }
        public Request Request { get; set; }

    }
}
```

Request.cs

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Net;
using static Universal_Tester.Utils.Enums;

namespace Universal_Tester.Entities
{
    public class Request : EntityBase
    {
        public string Url { get; set; }
        public List<Header> Headers { get; set; }
        [Display(Name = "Corpo da mensagem")]
        public string BodyMessage { get; set; }
        [Display(Name = "Conteúdo retornado")]
        public string ResultContent { get; set; }
        public MethodRequest MethodRequest { get; set; }
        public int AtributteValidateId { get; set; }
        public AttributeValidate AtributteValidate { get; set; }
        public int HttpStatusReturned { get; set; }
        public HttpStatusCode StatusCode { get; set; }
        public long TimeToExecute { get; set; }
        public string UserEmail { get; set; }
```

```
        public Request()
        {
            Headers = new List<Header>();
        }
    }
}
```

AccountViewModel.cs

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace Universal_Tester.Models
{
    public class ExternalLoginConfirmationViewModel
    {
        [Required]
        [Display(Name = "Email")]
        public string Email { get; set; }
    }

    public class ExternalLoginListViewModel
    {
        public string ReturnUrl { get; set; }
    }

    public class SendCodeViewModel
    {
        public string SelectedProvider { get; set; }
        public ICollection<System.Web.Mvc.SelectListItem> Providers { get; set; }
        public string ReturnUrl { get; set; }
        public bool RememberMe { get; set; }
    }

    public class VerifyCodeViewModel
    {
        [Required]
        public string Provider { get; set; }

        [Required]
        [Display(Name = "Code")]
        public string Code { get; set; }
        public string ReturnUrl { get; set; }

        [Display(Name = "Remember this browser?")]
        public bool RememberBrowser { get; set; }

        public bool RememberMe { get; set; }
    }

    public class ForgotViewModel
    {
        [Required]
        [Display(Name = "Email")]
```

```
        public string Email { get; set; }
    }

    public class LoginViewModel
    {
        [Required]
        [Display(Name = "Email")]
        [EmailAddress]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; set; }

        [Display(Name = "Remember me?")]
        public bool RememberMe { get; set; }
    }

    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "E-mail")]
        public string Email { get; set; }

        [Required]
        [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.",
MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "Senha")]
        public string Password { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirme sua senha")]
        [Compare("Password", ErrorMessage = "The password and confirmation
password do not match.")]
        public string ConfirmPassword { get; set; }
    }

    public class ResetPasswordViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "E-mail")]
        public string Email { get; set; }

        [Required]
        [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.",
MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "Senha")]
        public string Password { get; set; }
```

```
        [DataType(DataType.Password)]
        [Display(Name = "Confirm password")]
        [Compare("Password", ErrorMessage = "The password and confirmation
password do not match.")]
        public string ConfirmPassword { get; set; }

        public string Code { get; set; }
    }

    public class ForgotPasswordViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email")]
        public string Email { get; set; }
    }
}
```

IdentityModels.cs

```
using System.Data.Entity;
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;

namespace Universal_Tester.Models
{
    // You can add profile data for the user by adding more properties to your
ApplicationUser class, please visit https://go.microsoft.com/fwlink/?LinkID=317594 to
learn more.
    public class ApplicationUser : IdentityUser
    {
        public async Task<ClaimsIdentity>
GenerateUserIdentityAsync(UserManager<ApplicationUser> manager)
        {
            // Note the authenticationType must match the one defined in
CookieAuthenticationOptions.AuthenticationType
            var userIdentity = await manager.CreateIdentityAsync(this,
DefaultAuthenticationTypes.ApplicationCookie);
            // Add custom user claims here
            return userIdentity;
        }
    }

    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
    {
        public ApplicationDbContext()
            : base("DefaultConnection", throwIfV1Schema: false)
        {
        }

        public static ApplicationDbContext Create()
        {
```

```
            return new ApplicationDbContext();
        }
    }
}
```

ManageViewModels.cs

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNet.Identity;
using Microsoft.Owin.Security;

namespace Universal_Tester.Models
{
    public class IndexViewModel
    {
        public bool HasPassword { get; set; }
        public IList<UserLoginInfo> Logins { get; set; }
        public string PhoneNumber { get; set; }
        public bool TwoFactor { get; set; }
        public bool BrowserRemembered { get; set; }
    }

    public class ManageLoginsViewModel
    {
        public IList<UserLoginInfo> CurrentLogins { get; set; }
        public IList<AuthenticationDescription> OtherLogins { get; set; }
    }

    public class FactorViewModel
    {
        public string Purpose { get; set; }
    }

    public class SetPasswordViewModel
    {
        [Required]
        [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.",
MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "Nova senha")]
        public string NewPassword { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirme a nova senha")]
        [Compare("NewPassword", ErrorMessage = "The new password and confirmation
password do not match.")]
        public string ConfirmPassword { get; set; }
    }

    public class ChangePasswordViewModel
    {
        [Required]
        [DataType(DataType.Password)]
        [Display(Name = "Senha atual")]
```

```
        public string OldPassword { get; set; }

        [Required]
        [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.",
MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "Nova senha")]
        public string NewPassword { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirme a nova senha")]
        [Compare("NewPassword", ErrorMessage = "The new password and confirmation
password do not match.")]
        public string ConfirmPassword { get; set; }
    }

    public class AddPhoneNumberViewModel
    {
        [Required]
        [Phone]
        [Display(Name = "Phone Number")]
        public string Number { get; set; }
    }

    public class VerifyPhoneNumberViewModel
    {
        [Required]
        [Display(Name = "Code")]
        public string Code { get; set; }

        [Required]
        [Phone]
        [Display(Name = "Phone Number")]
        public string PhoneNumber { get; set; }
    }

    public class ConfigureTwoFactorViewModel
    {
        public string SelectedProvider { get; set; }
        public ICollection<System.Web.Mvc.SelectListItem> Providers { get; set; }
    }
}
```

RequestViewModel.cs

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Net;
using Universal_Tester.Entities;
using static Universal_Tester.Utils.Enums;

namespace Universal_Tester.Models
{
    public class RequestViewModel
```

```
    {
        public RequestViewModel()
        {
            Headers = new List<Header>
            {
                new Header("", ""),
                new Header("", ""),
                new Header("", "")
            };
        }

        public RequestViewModel(Request request)
        {
            id = request.Id;
            Url = request.Url;
            ValueReturned = request.AtributteValidate.ValueReturned;
            MethodRequest = request.MethodRequest;
            NameAttributeToValidate = request.AtributteValidate.NameAttributeToValidate;
            TypeOfAttribute = request.AtributteValidate.TypeOfAttribute;
            BodyMessage = request.BodyMessage;
            AtributteValidateId = request.AtributteValidateId;
            Headers = request.Headers;
            IsMatchValidation = request.AtributteValidate.IsMatchValidation;
            IsMatchTypeOfAttribute = request.AtributteValidate.IsMatchTypeOfAttribute;
        }

        public int id { get; set; }
        [Display(Name = "URL")]
        public string Url { get; set; }
        [Display(Name = "Headers", Description = "Conteúdo a ser enviado no corpo
requisição")]
        public List<Header> Headers { get; set; }
        [Display(Name = "Corpo da mensagem", Description = "Conteúdo a ser enviado
no corpo requisição")]
        public string BodyMessage { get; set; }
        [Display(Name = "Conteúdo retornado")]
        public string ResultContent { get; set; }
        [Display(Name = "Método", Description = "Método utilizado na requisição")]
        public MethodRequest MethodRequest { get; set; }
        public int AtributteValidateId { get; set; }
        [Display(Name = "Atributo a ser validado")]
        public string NameAttributeToValidate { get; set; }
        [Display(Name = "Está válido")]
        public bool IsMatchValidation { get; set; }
        [Display(Name = "Valor a ser validado no retorno")]
        public string ValueReturned { get; set; }
        [Display(Name = "Tipo de atributo")]
        public TypeOfAttribute TypeOfAttribute { get; set; }
        public int StatusCode { get; set; }
        public long TimeToExecute { get; set; }
        public bool IsMatchTypeOfAttribute { get; set; }

    }
}
```

RequestOperation.cs

```
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using RestSharp;
using System;
using System.Linq;
using Universal_Tester.Entities;
using static Universal_Tester.Utils.Enums;

namespace Universal_Tester.RequestServices
{
   public class RequestOperation
   {
      private IRestResponse response;
      private IRestRequest request;

      public string SendRequest(Request pRequest)
      {
         var restClient = new RestClient(pRequest.Url);
         request = new RestRequest(GetMethodType(pRequest.MethodRequest));
         if (pRequest.Headers != null && pRequest.Headers.Any())
            SetHeadersToRequest(pRequest);

         if (pRequest.MethodRequest != MethodRequest.GET &&
!string.IsNullOrWhiteSpace(pRequest.BodyMessage))
         {
            var objetoBody = JsonConvert.DeserializeObject(pRequest.BodyMessage);
            var bodyString = JsonConvert.SerializeObject(objetoBody);
            request.AddParameter("application/json", bodyString,
ParameterType.RequestBody);
         }
         response = restClient.Execute(request);
         ValidateKeyForCheck(response.Content, pRequest);
         pRequest.StatusCode = response.StatusCode;
         return response.Content;
      }
      private void SetHeadersToRequest(Request pRequest)
      {
         foreach (var header in pRequest.Headers)
            request.AddHeader(header.Key, header.Value);
      }

      private void ValidateKeyForCheck(string content, Request pRequest)
      {
         JObject objeto = new JObject();
         try
         {
            objeto = JObject.Parse(content);
         }
         catch(Exception e)
         {
            throw new Exception("Tipo de retorno diferente de JSONObject");
```

```
            }
            var attributeValueExpected =
objeto.SelectToken(pRequest.AtributteValidate.NameAttributeToValidate);
            SetIsMatchValidation(attributeValueExpected, pRequest);
            ValiteTypeAttribute(attributeValueExpected, pRequest);
        }
        private void SetIsMatchValidation(JToken attributeValueExpected, Request
pRequest)
        {
            if (attributeValueExpected == null)
            {
                pRequest.AtributteValidate.IsMatchValidation = false;
                pRequest.AtributteValidate.ValueReturned = null;
                pRequest.AtributteValidate.UpdatedAt = DateTime.Now;
            }
            else
            {
                pRequest.AtributteValidate.IsMatchValidation =
attributeValueExpected.Value<string>() == pRequest.AtributteValidate.ValueReturned;
                pRequest.AtributteValidate.ValueReturned =
attributeValueExpected.ToString();
                pRequest.AtributteValidate.UpdatedAt = DateTime.Now;
            }
        }
        private void ValiteTypeAttribute(JToken attributeValueExpected, Request
pRequest)
        {
            switch (pRequest.AtributteValidate.TypeOfAttribute)
            {
                case TypeOfAttribute.STRING:
                    ValidateTypeAttributeString(attributeValueExpected, pRequest);
                    break;
                case TypeOfAttribute.NUMBER:
                    ValidateTypeAttributeNumber(attributeValueExpected, pRequest);
                    break;
                case TypeOfAttribute.BOOLEAN:
                    ValidateTypeAttributeBool(attributeValueExpected, pRequest);
                    break;
                default:
                    ValidateTypeAttributeNull(attributeValueExpected, pRequest);
                    break;
            }
        }

        private void ValidateTypeAttributeNull(JToken attributeValueExpected, Request
pRequest)
        {
            try
            {
                pRequest.AtributteValidate.IsMatchTypeOfAttribute =
string.IsNullOrEmpty(attributeValueExpected.ToString());
            }
            catch (Exception e)
            {
                pRequest.AtributteValidate.IsMatchTypeOfAttribute = false;
```

```
        }
    }

    private void ValidateTypeAttributeBool(JToken attributeValueExpected, Request
pRequest)
    {
        try
        {
            var valorBooleano = bool.Parse(attributeValueExpected.ToString());
            pRequest.AtributteValidate.IsMatchTypeOfAttribute = valorBooleano;
        }
        catch (Exception e)
        {
            pRequest.AtributteValidate.IsMatchTypeOfAttribute = false;
        }
    }

    private void ValidateTypeAttributeString(JToken attributeValueExpected, Request
pRequest)
    {
        try
        {
            var valorString = attributeValueExpected.ToString();
            pRequest.AtributteValidate.IsMatchTypeOfAttribute = true;
        }
        catch (Exception e)
        {
            pRequest.AtributteValidate.IsMatchTypeOfAttribute = false;
        }
    }

    private void ValidateTypeAttributeNumber(JToken attributeValueExpected,
Request pRequest)
    {
        try
        {
            var valorString = decimal.Parse(attributeValueExpected.ToString());
            pRequest.AtributteValidate.IsMatchTypeOfAttribute = true;
        }
        catch (Exception e)
        {
            pRequest.AtributteValidate.IsMatchTypeOfAttribute = false;
        }
    }

    private Method GetMethodType(MethodRequest methodRequest)
    {
        switch (methodRequest)
        {
            case MethodRequest.GET:
                return Method.GET;
            case MethodRequest.POST:
                return Method.POST;
            case MethodRequest.PUT:
                return Method.PUT;
```

```
            default:
                return Method.DELETE;
        }
    }

  }
}
```

RequestService.cs

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Diagnostics;
using System.Linq;
using Universal_Tester.Context;
using Universal_Tester.Entities;
using Universal_Tester.Models;
using Universal_Tester.RequestServices;

namespace Universal_Tester.Services
{
   public class RequestsService
   {
      private UniversalTesterContext db = new UniversalTesterContext();
      private readonly RequestOperation requestOperation;
      public RequestsService()
      {
         requestOperation = new RequestOperation();
      }

      public List<Request> GetRequestsByUserEmail(string userEmail)
      {
         return db.Request.Include("AtributteValidate").Where(x => x.UserEmail ==
userEmail).ToList();
      }

      public RequestViewModel FindRequestToEditions(int requestId)
      {
         Request request = FindRequest(requestId);
         request.AtributteValidate =
db.AttributeValidate.Find(request.AtributteValidateId);
         request.Headers = db.Header.Where(x => x.RequestId == request.Id).ToList();
         return new RequestViewModel(request);
      }

      public void RemoveRequest(int id)
      {
         Request request = FindRequest(id);
         db.Request.Remove(request);
         db.SaveChanges();
      }
```

```
        public Request FindRequest(int id)
        {
            return db.Request.Find(id);
        }
        public void SaveEditionsRequest(RequestViewModel requestViewModel)
        {
            var request = FindRequest(requestViewModel.id);
            request.Url = requestViewModel.Url;
            request.Headers = requestViewModel.Headers.Where(x =>
!string.IsNullOrWhiteSpace(x.Key) && !string.IsNullOrWhiteSpace(x.Key)).ToList();
            request.MethodRequest = requestViewModel.MethodRequest;
            request.BodyMessage = requestViewModel.BodyMessage;
            request.UpdatedAt = DateTime.Now;
            var attributteValidate = FindAttributeToValidate(requestViewModel.id);
            attributteValidate.ValueReturned = requestViewModel.ValueReturned;
            attributteValidate.NameAttributeToValidate =
requestViewModel.NameAttributeToValidate;
            attributteValidate.TypeOfAttribute = requestViewModel.TypeOfAttribute;
            attributteValidate.UpdatedAt = DateTime.Now;
            attributteValidate.IsMatchTypeOfAttribute =
requestViewModel.IsMatchTypeOfAttribute;
            attributteValidate.IsMatchValidation = requestViewModel.IsMatchValidation;
            db.Entry(attributteValidate).State = EntityState.Modified;
            db.Entry(request).State = EntityState.Modified;
            db.SaveChanges();
        }
        public AttributeValidate FindAttributeToValidate(int id)
        {
            return db.AttributeValidate.Find(id);
        }

        public Request AddRequest(Request request)
        {
            return db.Request.Add(request);
        }

        public void CreateRequest(RequestViewModel requestViewModel, string
userEmail)
        {
            var request = SendRequest(requestViewModel);
            request.UserEmail = userEmail;
            var headers = request.Headers;
            request.Headers = null;
            AddRequest(request);
            db.SaveChanges();
            foreach (var item in headers)
            {
                item.RequestId = request.Id;
                item.CreatedAt = DateTime.Now;
                item.UpdatedAt = DateTime.Now;
                db.Header.Add(item);
            }
            db.SaveChanges();
        }
```

```
        public Request SendRequest(RequestViewModel requestViewModel)
        {
            Stopwatch stopWatch = new Stopwatch();
            stopWatch.Start();
            var request = new Request()
            {
                Url = requestViewModel.Url,
                BodyMessage = requestViewModel.BodyMessage,
                AtributteValidate = new AttributeValidate(),
                MethodRequest = requestViewModel.MethodRequest,
                CreatedAt = DateTime.Now,
                UpdatedAt = DateTime.Now,
            };
            SetHeaders(request, requestViewModel);
            SetAttributeValidate(request, requestViewModel);
            request.ResultContent = requestOperation.SendRequest(request);
            requestViewModel.ResultContent = request.ResultContent;
            stopWatch.Stop();
            request.TimeToExecute = stopWatch.ElapsedMilliseconds;
            return request;
        }
        private void SetHeaders(Request request, RequestViewModel requestViewModel)
        {
            request.Headers = requestViewModel.Headers.Where(x =>
!string.IsNullOrWhiteSpace(x.Key)
                                                && !string.IsNullOrWhiteSpace(x.Value))
                                                .Select(x => new Header(x.Key, x.Value))
                                                .ToList();
        }
        private void SetAttributeValidate(Request request, RequestViewModel
requestViewModel)
        {
            request.AtributteValidate.NameAttributeToValidate =
requestViewModel.NameAttributeToValidate;
            request.AtributteValidate.TypeOfAttribute = requestViewModel.TypeOfAttribute;
            request.AtributteValidate.ValueReturned = requestViewModel.ValueReturned;
            request.AtributteValidate.CreatedAt = DateTime.Now;
        }

        public void ExecuteAll(List<Request> requestList)
        {
            requestList.ForEach(x =>
            {
                Stopwatch stopWatch = new Stopwatch();
                stopWatch.Start();
                requestOperation.SendRequest(x);
                x.CreatedAt = DateTime.Now;
                x.UpdatedAt = DateTime.Now;
                stopWatch.Stop();
                x.TimeToExecute = stopWatch.ElapsedMilliseconds;
                db.Request.Add(x);
            });
            db.SaveChanges();
        }
    }
```

```
}
```

Enums.cs

```
namespace Universal_Tester.Utils
{
    public class Enums
    {

        public enum MethodRequest
        {
            GET = 0,
            POST = 1,
            PUT = 2,
            DELETE = 3,
        }

        public enum TypeOfAttribute
        {
            STRING = 1,
            BOOLEAN = 2,
            NUMBER = 3,
            NULL = 4
        }

    }
}
```

UniversalDbContext.cs

```
using Microsoft.AspNet.Identity.EntityFramework;
using System.Data.Entity;
using Universal_Tester.Entities;
using Universal_Tester.Models;

namespace Universal_Tester.Context
{
    public class UniversalTesterContext : IdentityDbContext<ApplicationUser>
    {
        public UniversalTesterContext() : base("DefaultConnection")
        {
        }

        public static UniversalTesterContext Create()
        {
            return new UniversalTesterContext();
        }

        public DbSet<Request> Request { get; set; }
        public DbSet<Header> Header { get; set; }
        public DbSet<AttributeValidate> AttributeValidate { get; set; }

    }
```

```
}
```

CreateDataBase.cs

```
namespace Universal_Tester.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class CreateDataBase : DbMigration
    {
        public override void Up()
        {
            CreateTable(
                "dbo.AttributeValidates",
                c => new
                    {
                        Id = c.Int(nullable: false, identity: true),
                        NameAttributeToValidate = c.String(),
                        IsMatchValidation = c.Boolean(nullable: false),
                        ValueReturned = c.String(),
                        TypeOfAttribute = c.Int(nullable: false),
                        IsMatchTypeOfAttribute = c.Boolean(nullable: false),
                        CreatedAt = c.DateTime(nullable: false),
                        UpdatedAt = c.DateTime(nullable: false),
                    })
                .PrimaryKey(t => t.Id);

            CreateTable(
                "dbo.Headers",
                c => new
                    {
                        Id = c.Int(nullable: false, identity: true),
                        Key = c.String(),
                        Value = c.String(),
                        Description = c.String(),
                        CreatedAt = c.DateTime(nullable: false),
                        UpdatedAt = c.DateTime(nullable: false),
                        Request_Id = c.Int(),
                    })
                .PrimaryKey(t => t.Id)
                .ForeignKey("dbo.Requests", t => t.Request_Id)
                .Index(t => t.Request_Id);

            CreateTable(
                "dbo.Requests",
                c => new
                    {
                        Id = c.Int(nullable: false, identity: true),
                        Url = c.String(),
                        BodyMessage = c.String(),
                        ResultContent = c.String(),
                        MethodRequest = c.Int(nullable: false),
                        AtributteValidateId = c.Int(nullable: false),
                        CreatedAt = c.DateTime(nullable: false),
```

```
                UpdatedAt = c.DateTime(nullable: false),
            })
        .PrimaryKey(t => t.Id)
        .ForeignKey("dbo.AttributeValidates", t => t.AtributteValidateId,
cascadeDelete: true)
        .Index(t => t.AtributteValidateId);

    CreateTable(
        "dbo.AspNetRoles",
        c => new
            {
                Id = c.String(nullable: false, maxLength: 128),
                Name = c.String(nullable: false, maxLength: 256),
            })
        .PrimaryKey(t => t.Id)
        .Index(t => t.Name, unique: true, name: "RoleNameIndex");

    CreateTable(
        "dbo.AspNetUserRoles",
        c => new
            {
                UserId = c.String(nullable: false, maxLength: 128),
                RoleId = c.String(nullable: false, maxLength: 128),
            })
        .PrimaryKey(t => new { t.UserId, t.RoleId })
        .ForeignKey("dbo.AspNetRoles", t => t.RoleId, cascadeDelete: true)
        .ForeignKey("dbo.AspNetUsers", t => t.UserId, cascadeDelete: true)
        .Index(t => t.UserId)
        .Index(t => t.RoleId);

    CreateTable(
        "dbo.AspNetUsers",
        c => new
            {
                Id = c.String(nullable: false, maxLength: 128),
                Email = c.String(maxLength: 256),
                EmailConfirmed = c.Boolean(nullable: false),
                PasswordHash = c.String(),
                SecurityStamp = c.String(),
                PhoneNumber = c.String(),
                PhoneNumberConfirmed = c.Boolean(nullable: false),
                TwoFactorEnabled = c.Boolean(nullable: false),
                LockoutEndDateUtc = c.DateTime(),
                LockoutEnabled = c.Boolean(nullable: false),
                AccessFailedCount = c.Int(nullable: false),
                UserName = c.String(nullable: false, maxLength: 256),
            })
        .PrimaryKey(t => t.Id)
        .Index(t => t.UserName, unique: true, name: "UserNameIndex");

    CreateTable(
        "dbo.AspNetUserClaims",
        c => new
            {
                Id = c.Int(nullable: false, identity: true),
```

```
                UserId = c.String(nullable: false, maxLength: 128),
                ClaimType = c.String(),
                ClaimValue = c.String(),
            })
        .PrimaryKey(t => t.Id)
        .ForeignKey("dbo.AspNetUsers", t => t.UserId, cascadeDelete: true)
        .Index(t => t.UserId);

      CreateTable(
        "dbo.AspNetUserLogins",
        c => new
            {
                LoginProvider = c.String(nullable: false, maxLength: 128),
                ProviderKey = c.String(nullable: false, maxLength: 128),
                UserId = c.String(nullable: false, maxLength: 128),
            })
        .PrimaryKey(t => new { t.LoginProvider, t.ProviderKey, t.UserId })
        .ForeignKey("dbo.AspNetUsers", t => t.UserId, cascadeDelete: true)
        .Index(t => t.UserId);

    }

    public override void Down()
    {
        DropForeignKey("dbo.AspNetUserRoles", "UserId", "dbo.AspNetUsers");
        DropForeignKey("dbo.AspNetUserLogins", "UserId", "dbo.AspNetUsers");
        DropForeignKey("dbo.AspNetUserClaims", "UserId", "dbo.AspNetUsers");
        DropForeignKey("dbo.AspNetUserRoles", "RoleId", "dbo.AspNetRoles");
        DropForeignKey("dbo.Headers", "Request_Id", "dbo.Requests");
        DropForeignKey("dbo.Requests", "AtributteValidateId",
"dbo.AttributeValidates");
        DropIndex("dbo.AspNetUserLogins", new[] { "UserId" });
        DropIndex("dbo.AspNetUserClaims", new[] { "UserId" });
        DropIndex("dbo.AspNetUsers", "UserNameIndex");
        DropIndex("dbo.AspNetUserRoles", new[] { "RoleId" });
        DropIndex("dbo.AspNetUserRoles", new[] { "UserId" });
        DropIndex("dbo.AspNetRoles", "RoleNameIndex");
        DropIndex("dbo.Requests", new[] { "AtributteValidateId" });
        DropIndex("dbo.Headers", new[] { "Request_Id" });
        DropTable("dbo.AspNetUserLogins");
        DropTable("dbo.AspNetUserClaims");
        DropTable("dbo.AspNetUsers");
        DropTable("dbo.AspNetUserRoles");
        DropTable("dbo.AspNetRoles");
        DropTable("dbo.Requests");
        DropTable("dbo.Headers");
        DropTable("dbo.AttributeValidates");
    }
  }
}
```

CreateColumn_StatusCode.cs

```
namespace Universal_Tester.Migrations
{
```

```
    using System;
    using System.Data.Entity.Migrations;

    public partial class CreateColumn_StatusCode : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.Requests", "HttpStatusReturned", c => c.Int(nullable: false));
            AddColumn("dbo.Requests", "StatusCode", c => c.Int(nullable: false));
        }

        public override void Down()
        {
            DropColumn("dbo.Requests", "StatusCode");
            DropColumn("dbo.Requests", "HttpStatusReturned");
        }
    }
}
```

AddColumnTimeToExecute.cs

```
namespace Universal_Tester.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddColumnTimeToExecute : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.Requests", "TimeToExecute", c => c.Long(nullable: false));
        }

        public override void Down()
        {
            DropColumn("dbo.Requests", "TimeToExecute");
        }
    }
}
```

AddColumnUserEmail.cs

```
namespace Universal_Tester.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddColumnUserEmail : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.Requests", "UserEmail", c => c.String());
        }
```

```
        public override void Down()
        {
            DropColumn("dbo.Requests", "UserEmail");
        }
    }
}
```

RemoveColumnDescription.cs

```
namespace Universal_Tester.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class RemoveColumnDescription : DbMigration
    {
        public override void Up()
        {
            DropColumn("dbo.Headers", "Description");
        }

        public override void Down()
        {
            AddColumn("dbo.Headers", "Description", c => c.String());
        }
    }
}
```

AddForeignKey_RequestId.cs

```
namespace Universal_Tester.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddForeignKey_RequestId : DbMigration
    {
        public override void Up()
        {
            DropForeignKey("dbo.Headers", "Request_Id", "dbo.Requests");
            DropIndex("dbo.Headers", new[] { "Request_Id" });
            RenameColumn(table: "dbo.Headers", name: "Request_Id", newName:
"RequestId");
            AlterColumn("dbo.Headers", "RequestId", c => c.Int(nullable: false));
            CreateIndex("dbo.Headers", "RequestId");
            AddForeignKey("dbo.Headers", "RequestId", "dbo.Requests", "Id",
cascadeDelete: true);
        }

        public override void Down()
        {
            DropForeignKey("dbo.Headers", "RequestId", "dbo.Requests");
            DropIndex("dbo.Headers", new[] { "RequestId" });
            AlterColumn("dbo.Headers", "RequestId", c => c.Int());
```

```
            RenameColumn(table: "dbo.Headers", name: "RequestId", newName:
"Request_Id");
            CreateIndex("dbo.Headers", "Request_Id");
            AddForeignKey("dbo.Headers", "Request_Id", "dbo.Requests", "Id");
        }
    }
}
```

Create.cshtml

```
@model Universal_Tester.Models.RequestViewModel

@{
    ViewBag.Title = "Nova Requisição";
}


<style>
    textarea {
        margin: 0px;
        width: 411px;
        height: 157px;
    }
</style>

<h2>NOVA REQUISIÇÃO</h2>


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">

        <hr />


        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.Url, htmlAttributes: new { @class = "control-
label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Url, new { htmlAttributes = new { @class =
"form-control" } })
                @Html.ValidationMessageFor(model => model.Url, "", new { @class = "text-
danger" })
            </div>
        </div>

        <div class="form-group" id="labelHeaders" hidden>
            @Html.LabelFor(model => model.Headers, htmlAttributes: new { @class =
"control-label col-md-2" })
        </div>
```

```
        <div id="Headers" style="margin-left: 183px;" hidden>
            @for (int i = 0; i < Model.Headers.Count; i++)
            {
                <div class="form-group">
                    @Html.LabelFor(model => model.Headers[i].Key, htmlAttributes: new {
@class = "control-label col-md-1" })
                    <div class="col-md-3">
                        @Html.EditorFor(model => model.Headers[i].Key, new { htmlAttributes
= new { @class = "form-control" } })
                    </div>
                    @Html.LabelFor(model => model.Headers[i].Value, htmlAttributes: new {
@class = "control-label col-md-1" })
                    <div class="col-md-3">
                        @Html.EditorFor(model => model.Headers[i].Value, new {
htmlAttributes = new { @class = "form-control" } })
                    </div>
                </div>
            }
        </div>


        <div class="form-group">
            @Html.LabelFor(model => model.MethodRequest, htmlAttributes: new {
@class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EnumDropDownListFor(model => model.MethodRequest,
htmlAttributes: new { @class = "form-control" })
                @Html.ValidationMessageFor(model => model.MethodRequest, "", new {
@class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.NameAttributeToValidate, htmlAttributes: new
{ @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.NameAttributeToValidate, new {
htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.NameAttributeToValidate, "",
new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.TypeOfAttribute, htmlAttributes: new {
@class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EnumDropDownListFor(model => model.TypeOfAttribute,
htmlAttributes: new { @class = "form-control" })
                @Html.ValidationMessageFor(model => model.BodyMessage, "", new {
@class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
```

```
            @Html.LabelFor(model => model.ValueReturned, htmlAttributes: new { @class
= "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.ValueReturned, new { htmlAttributes =
new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.ValueReturned, "", new {
@class = "text-danger" })
            </div>
        </div>

        <div id="bodyMessage" hidden>
            <div class="form-group">
                @Html.LabelFor(model => model.BodyMessage, htmlAttributes: new {
@class = "control-label col-md-2" })
                <div class="col-md-10">
                    @Html.TextAreaFor(model => model.BodyMessage, new { htmlAttributes
= new { @class = "form-control textarea" } })
                    @Html.ValidationMessageFor(model => model.BodyMessage, "", new {
@class = "text-danger" })
                </div>
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </div>
    </div>
}

<div>
    @Html.ActionLink("Ir para a listagem", "Index")
</div>

<script src="~/Scripts/jquery-1.10.2.min.js"></script>
<script>

    $(document).ready(function () {
        console.log("ready!");
    });

    $("#MethodRequest").change(function () {
        var valorMethodRequest = $("#MethodRequest option:selected").val();
        if (valorMethodRequest > 0) {
            $('#bodyMessage').show();
            $('#labelHeaders').show();
            $('#Headers').show();
        } else {
            $('#labelHeaders').hide();
            $('#bodyMessage').hide();
            $('#Headers').hide();
        }
    });
```

```
        //$("#add").click(function () {
        //    debugger;
        //    var element = $('#Headers');
        //    element.append('<div class="form-group"> <label class="control-label col-md-
2" for="Header" > Header</label> <div class="col-md-3">  <input class="form-control
text-box single-line" id="Url" name= "Url" type="text" value="" > </div> <div class="col-
md-3"> <input class="form-control text-box single-line" id="Url" name="Url" type="text"
value=""> </div > </div>');
        //});

</script>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Delete.cshtml

```
@model Universal_Tester.Entities.Request

@{
    ViewBag.Title = "Delete";
}

<h2>Delete</h2>

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Request</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model =>
model.AtributteValidate.NameAttributeToValidate)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.AtributteValidate.NameAttributeToValidate)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Url)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Url)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.BodyMessage)
        </dt>

        <dd>
```

```
            @Html.DisplayFor(model => model.BodyMessage)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.ResultContent)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.ResultContent)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.MethodRequest)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.MethodRequest)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.CreatedAt)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.CreatedAt)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.UpdatedAt)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.UpdatedAt)
        </dd>

    </dl>

    @using (Html.BeginForm()) {
        @Html.AntiForgeryToken()

        <div class="form-actions no-color">
            <input type="submit" value="Delete" class="btn btn-default" /> |
            @Html.ActionLink("Back to List", "Index")
        </div>
    }
</div>
```

Details.cshtml

```
@model Universal_Tester.Entities.Request

@{
    ViewBag.Title = "Details";
```

```
}

<h2>Details</h2>

<div>
    <h4>Request</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model =>
model.AtributteValidate.NameAttributeToValidate)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.AtributteValidate.NameAttributeToValidate)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Url)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Url)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.BodyMessage)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.BodyMessage)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.ResultContent)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.ResultContent)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.MethodRequest)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.MethodRequest)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.CreatedAt)
        </dt>

        <dd>
```

```
            @Html.DisplayFor(model => model.CreatedAt)
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.UpdatedAt)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.UpdatedAt)
        </dd>

    </dl>
</div>
<p>
    @Html.ActionLink("Edit", "Edit", new { id = Model.Id }) |
    @Html.ActionLink("Back to List", "Index")
</p>
```

Edit.cshtml

```
@model Universal_Tester.Models.RequestViewModel

@{
    ViewBag.Title = "Edit";
}

<h2>Edit</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <input Name="id" name="id" id="id" type="hidden" value="@Model.id" />
        <div class="form-group">
            @Html.LabelFor(model => model.Url, htmlAttributes: new { @class = "control-
label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Url, new { htmlAttributes = new { @class =
"form-control" } })
                @Html.ValidationMessageFor(model => model.Url, "", new { @class = "text-
danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.MethodRequest, htmlAttributes: new {
@class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EnumDropDownListFor(model => model.MethodRequest,
htmlAttributes: new { @class = "form-control" })
```

```
            @Html.ValidationMessageFor(model => model.MethodRequest, "", new {
@class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.NameAttributeToValidate, htmlAttributes: new
{ @class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.NameAttributeToValidate, new {
htmlAttributes = new { @class = "form-control" } })
            @Html.ValidationMessageFor(model => model.NameAttributeToValidate, "",
new { @class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.TypeOfAttribute, htmlAttributes: new {
@class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EnumDropDownListFor(model => model.TypeOfAttribute,
htmlAttributes: new { @class = "form-control" })
            @Html.ValidationMessageFor(model => model.BodyMessage, "", new {
@class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.ValueReturned, htmlAttributes: new { @class
= "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EditorFor(model => model.ValueReturned, new { htmlAttributes =
new { @class = "form-control" } })
            @Html.ValidationMessageFor(model => model.ValueReturned, "", new {
@class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.TypeOfAttribute, htmlAttributes: new {
@class = "control-label col-md-2" })
        <div class="col-md-10">
            @Html.EnumDropDownListFor(model => model.TypeOfAttribute,
htmlAttributes: new { @class = "form-control" })
            @Html.ValidationMessageFor(model => model.BodyMessage, "", new {
@class = "text-danger" })
        </div>
    </div>

    <div class="form-group">
        @Html.LabelFor(model => model.BodyMessage, htmlAttributes: new { @class
= "control-label col-md-2" })
        <div class="col-md-10">
```

```
            @Html.EditorFor(model => model.BodyMessage, new { htmlAttributes = new
{ @class = "form-control" } })
            @Html.ValidationMessageFor(model => model.BodyMessage, "", new {
@class = "text-danger" })
        </div>
    </div>


    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <input type="submit" value="Salvar Edições" class="btn btn-default" />
        </div>
    </div>
</div>
}


<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Index.cshtml

```
@model IEnumerable<Universal_Tester.Models.RequestViewModel>

@{
    ViewBag.Title = "Index";
}

<h2>Seu histórico de testes e resultados</h2>

<p>
    @Html.ActionLink("Nova requisição", "Create")
</p>
<table class="table">
    <tr>
        <th>
            Nome do atributo
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Url)
        </th>
        <th>
            Valor Retornado
        </th>
        <th>
            Método
        </th>
        <th>
```

```
        Valor Correto
    </th>
    <th>
        Atributo correto
    </th>
    <th>
        Tipo de atributo retornado
    </th>
    <th>
        Status HTTP
    </th>
    <th>
        Duração (Milissegundos)
    </th>
    <th>Ações</th>
</tr>

@*<tr>
    <th>
        @Html.DisplayNameFor(model => model.NameAttributeToValidate)
    </th>
    <th>
        @Html.DisplayNameFor(model => model.Url)
    </th>
    <th>
        @Html.DisplayNameFor(model => model.ValueReturned)
    </th>
    <th>
        @Html.DisplayNameFor(model => model.MethodRequest)
    </th>
    <th>
        @Html.DisplayNameFor(model => model.IsMatchValidation)
    </th>
    <th>
        @Html.DisplayNameFor(model => model.IsMatchTypeOfAttribute)
    </th>
    <th>
        @Html.DisplayNameFor(model => model.TypeOfAttribute)
    </th>
    <th></th>
</tr>*@




@foreach (var item in Model)
{
<tr>
    <td>
        @Html.DisplayFor(modelItem => item.NameAttributeToValidate)
    </td>
    <td>
        @Html.DisplayFor(modelItem => item.Url)
    </td>
    <td>
```

```
            @Html.DisplayFor(modelItem => item.ValueReturned)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.MethodRequest)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.IsMatchValidation)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.IsMatchTypeOfAttribute)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.TypeOfAttribute)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.StatusCode)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.TimeToExecute)
        </td>
        <td>
            @Html.ActionLink("Editar", "Edit", new { id = item.id })
            @Html.ActionLink("Remover", "Delete", new { id = item.id })
        </td>
    </tr>
    }

</table>

@Html.ActionLink("Reexecutar Todos", "ExecuteAll")
```

Layout.cshtml

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Universal Tester</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")

</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target=".navbar-collapse">
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
```

```
            @*@Html.ActionLink("INÍCIO", "Index", "Home", new { area = "" }, new {
@class = "navbar-brand" })*@
        </div>
        <div class="navbar-collapse collapse">
            @{ if (User.Identity.IsAuthenticated)
                {
                    <ul class="nav navbar-nav">
                        @*<li>@Html.ActionLink("Home", "Index", "Home")</li>
                        <li>@Html.ActionLink("About", "About", "Home")</li>*@
                        <li>@Html.ActionLink("Nova Requisição", "Create", "Request")</li>
                    </ul>
                }
            }
            @Html.Partial("_LoginPartial")
        </div>
    </div>
</div>
<div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; @DateTime.Now.Year - Testador universal de requisições</p>
    </footer>
</div>

@Scripts.Render("~/bundles/jquery")
@Scripts.Render("~/bundles/bootstrap")
@RenderSection("scripts", required: false)
</body>
</html>
```

LoginPartial.cshtml

```
@using Microsoft.AspNet.Identity
@if (Request.IsAuthenticated)
{
    using (Html.BeginForm("LogOff", "Account", FormMethod.Post, new { id =
"logoutForm", @class = "navbar-right" }))
    {
    @Html.AntiForgeryToken()

    <ul class="nav navbar-nav navbar-right">
        <li>
            @Html.ActionLink("Logado como: " + User.Identity.GetUserName() + "!",
"Index", "Manage", routeValues: null, htmlAttributes: new { title = "Manage" })
        </li>
        <li><a href="javascript:document.getElementById('logoutForm').submit()">Log
off</a></li>
    </ul>
    }
}
else
{
    <ul class="nav navbar-nav navbar-right">
```

```
        <li>@Html.ActionLink("Registrar-se", "Register", "Account", routeValues: null,
htmlAttributes: new { id = "registerLink" })</li>
        <li>@Html.ActionLink("Fazer Login", "Login", "Account", routeValues: null,
htmlAttributes: new { id = "loginLink" })</li>
    </ul>
}
```

Error.cshtml

```
@model System.Web.Mvc.HandleErrorInfo

@{
    ViewBag.Title = "Error";
}

<h1 class="text-danger">Error.</h1>
<h2 class="text-danger">Erro ocorrido ao processar a request.</h2>
```

Lockout.cshtml

```
@model System.Web.Mvc.HandleErrorInfo

@{
    ViewBag.Title = "Locked Out";
}

<hgroup>
    <h1 class="text-danger">Locked out.</h1>
    <h2 class="text-danger">Conta bloqueada para acesso, favor tentar mais
tarde.</h2>
</hgroup>
```